



Engineers' Guide to Embedded Linux & Android

Yocto Project – Big in Embedded Linux

Is Device Security a Major Challenge for
Open Source Software?

Android Dives into Embedded Despite
Fragmentation Risks

Annual Industry Guide

Solutions for engineers and embedded developers
using Embedded Linux and Android

EECatalog www.eecatalog.com/embeddedlinux

Featured Products



Enea Linux PlatformBuilder



From TeamF1: SecureF1st
Network Attached
Storage Solution



From EMAC: PPC-E7+

Platinum Sponsor



Gold Sponsors



Leading Embedded Development Tools...



DS-5™ Development Tools for ARM

- Professional software development solution
- Powerful debugger for bring-up, driver and application development
- Graphical performance analysis tools
- Compiler and fast ARM processor-based SoC simulation models

www.arm.com/ds5
1-800-348-8051



YOUR LINUX, YOUR WAY



Enea® Linux PlatformBuilder is an easy-to-use cost-effective offering that lets you quickly configure, build, test, debug and maintain application and platform software for single core, modern multi CPU and multicore devices based on the Linux operating system.

www.enea.com/elpb

Enea® and Enea® Linux PlatformBuilder are trademarks of Enea AB and its subsidiaries. Any other company, product or service names mentioned above are registered or unregistered trademarks of their respective owner. © Enea AB 2011.



Welcome to the Engineers' Guide to Embedded Linux and Android 2012



Linux in its many iterations – including Android – continues to make headway in embedded designs. In fact, growth is expected to maintain double-digit increases through 2013, according to VDC Research. This issue is chock-full of information to help you keep up.

In “Yocto Project – Big in Embedded Linux,” The Linux Foundation explains the advantages of Linux and the Yocto open collaboration that provides tools and infrastructure to help create custom Linux-based systems for embedded products regardless of the hardware architecture. Throughout this issue, key vendors provide detailed insight into some of the most important considerations for embedded Linux designs, including multicore development, accessory design, and device security. See the Mentor Graphics article “How Do You Eat a Multicore Elephant?”; Microchip’s “Utilizing Google’s New Open Accessory Framework for Android Accessory Designs”; and Wind River’s “Is Device Security a Major Challenge for Open Source Software?” We also look at some interesting trends in modeling open source design in a discussion with Embedded Market Forecaster’s Jerry Krasner. And our roundtable discussion covers development challenges and opportunities, security and the ups and downs of Android in embedded applications. Of course, you’ll also find lots more information in these pages on the products and services you need to succeed in your embedded Android and Linux applications.

We hope you enjoy this EE Catalog Resource Guide. As always, we’d love to hear your feedback, thoughts and comments. Send them to info@extensionmedia.com.



Cheryl Berglund Coupé

Editor

P.S. To subscribe to our series of Engineers’ Guides for embedded developers and engineers, visit:

www.eecatalog.com/subscribe

Engineers’ Guide to Embedded Linux and Android 2012

www.eecatalog.com/embeddedlinux

VP/Associate Publisher

Clair Bright
cbright@extensionmedia.com
(415) 255-0390 ext. 15

Editorial

Editorial Director

John Blyler
jblyler@extensionmedia.com
(503) 614-1082

Editor

Cheryl Coupé
ccoupe@extensionmedia.com

Creative/Production

Production Manager

Spryte Heithecker

Graphic Designers

Keith Kelly - Senior
Nicky Jacobson

Production Assistant

Jenn Burkhardt

Senior Web Developer

Mariam Moattari

Advertising/Reprint Sales

VP/Associate Publisher

Embedded Electronics Media Group
Clair Bright
cbright@extensionmedia.com
(415) 255-0390 ext. 15

Sales Manager

Marcy Carnerie
mcarnerie@extensionmedia.com
(510) 919-4788

Marketing/Circulation

Jenna Johnson

To Subscribe

www.extensionmedia.com/free

Extension

MEDIA

Extension Media, LLC Corporate Office

President and Publisher

Vince Ridley
vridley@extensionmedia.com

Vice President, Sales

Embedded Electronics Media Group
Clair Bright
cbright@extensionmedia.com

Vice President, Marketing and Product Development

Karen Murray
kmurray@extensionmedia.com

Vice President, Business Development

Melissa Sterling
msterling@extensionmedia.com

Special Thanks to Our Sponsors

montavista™ ARM ENEA

TeamF1 WIND RIVER

The Engineers' Guide to Embedded Linux and Android is published by Extension Media LLC. Extension Media makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this Catalog nor does it make a commitment to update the information contained herein. Engineers' Guide to Embedded Linux and Android is Copyright © 2011 Extension Media LLC. No information in this Catalog may be reproduced without expressed written permission from Extension Media @ 1786 18th Street, San Francisco, CA 94107-2343.

All registered trademarks and trademarks included in this Catalog are held by their respective companies. Every attempt was made to include all trademarks and registered trademarks where indicated by their companies.

Developing a home
or business router?



Here's the key to help it stand out
from the competition...



Field-proven, high-performance security software from TeamF1

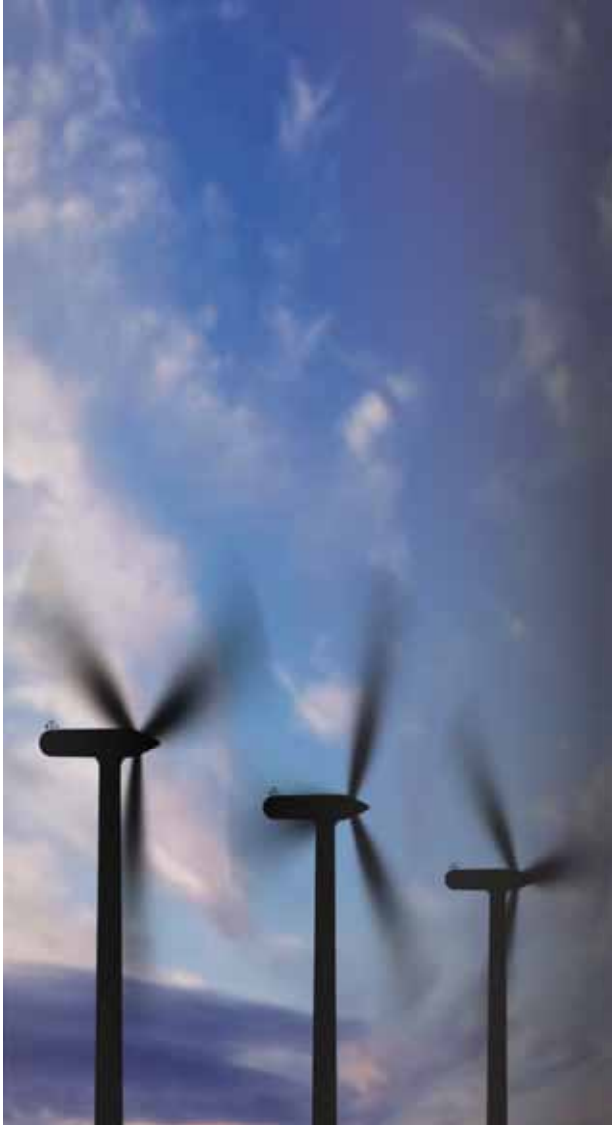
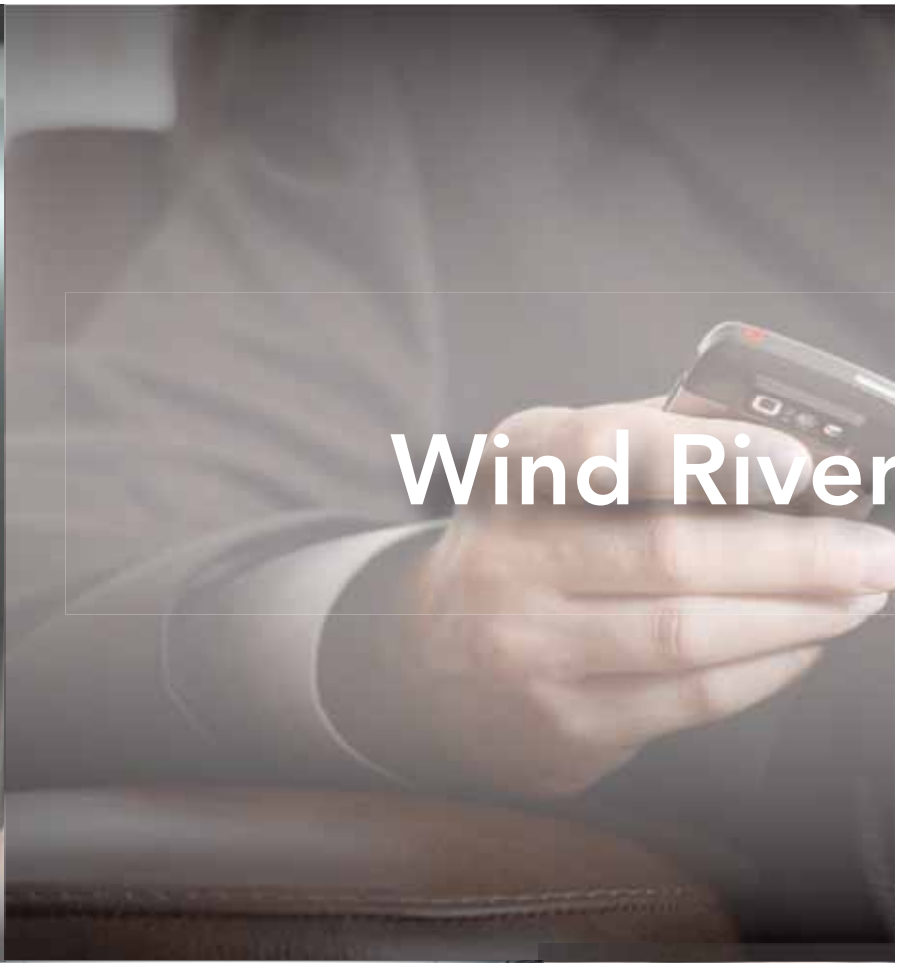
For more than a decade, TeamF1's technologies have helped connect and secure embedded devices in various markets. Our software has enabled networking OEMs and service providers to bring next-generation home/SOHO/SMB gateway devices to the market faster and with far less risk than otherwise possible.

Not only that - TeamF1's production-ready software is delivered as a "customized standard product" tailored to your requirements to help you differentiate your product from your competitors'. With an emphasis on higher performance, superior security, and advanced networking technologies, we can help enable innovation in your home and business routers. Fast. Under-budget. Now.

Contact us to learn how you can leverage TeamF1's offering and make your device stand out from the crowd. After all, the help key is not just part of our name but also our mission.

CPE Routers | VPN / Firewalls | UTM Gateways







Salutes 20 Years of Linux

- 1981 Wind River is founded.
- 1991 Linux is born.
- 2004 Wind River releases Wind River Linux 1.0.
- 2009 Wind River is the No. 1 commercial embedded Linux.
- 2010 Wind River releases Wind River Linux 4.
- 2010 Wind River joins the Yocto Project.
- 2011 Wind River releases Wind River Linux Secure.
- 2011 Linux turns 20.
- 2011 Wind River announces Wind River Linux Graphics Release

Thursday, August 18 at 10:30 am

User Space Tracing in Small Footprint Devices (How Low Can You Go?)
Jason Wessel, Principal Technologist, Wind River

Friday, August 19 at 10:15 am

Where is the Money in Open Source?
Business Models and the Marketing of Open Source Technologies
Nithya Ruff, Senior Director of Linux Product Marketing, Wind River

WIND RIVER

Contents

No Free Lunch with Embedded Operating Systems

By Cheryl Coupé..... 8

Android Dives into Embedded Despite Fragmentation Risks

By Cheryl Coupé..... 9

Yocto Project – Big in Embedded Linux

By Rudolf Streif, director of embedded solutions, The Linux Foundation..... 13

From Zero to Boot: Porting Android to your ARM platform

By Vassilis Laganakos, senior embedded software engineer, ARM Ltd. 17

How Do You Eat a Multicore Elephant?

By Stephen Olsen, Mentor Graphics..... 20

Utilizing Google's New Open Accessory Framework for Android™ Accessory Designs

By David Flowers, principal applications engineer, Advanced Microcontroller Architecture Division, Microchip Technology Inc...... 23

Crunching the Numbers: Sensors as Co-Processors in Next-Gen Smartphones?

By Ed Brachocki, Kionix & Stephen Maine, Opta3 llc 27

Online & Offline → Embedded Linux and Android Resources 30

Is Device Security a Major Challenge for Open Source Software?

By Milind Kukanur, senior product manager, Wind River 40

Products and Services

Hardware / Hardware Tools

Boards

EMAC, Inc

PPC-E7+.....31

Software Products

Software Development Tools

Mentor Embedded Linux Solutions

The Mentor approach streamlines platform creation and application development – allowing you to focus on value-add contributions.....32

Operating Systems / Distributions

Enea

Enea Linux PlatformBuilder 33

Networking / Communication Libraries

TeamF1, Inc.

SecureF1rst CPE Gateway Solution 35

Networking / Communication Packages

TeamF1, Inc.

SecureF1rst Network Attached Storage Solution.....36

SecureF1rst Security Gateway Solution 37

Services

Training

The PTR Group

Embedded Linux and Android Training Classes 39

Lowest Power Consumption Most Powerful Performance

Energy Saving and Performance Enhancing can Coexist

New Generation SoC Design

- Fanless Design
- Built-in MTBF Flag Counter
- Onboard 4MB SPI Flash Disk
- Support Floating Point Unit
- Support PCI / ISA / SPI Bus
- Wide Operating Temp. -40 ~+85 °C

VDX-6354

Embedded Vortex86DX PC/104 CPU Module

- Vortex86DX CPU 800MHz
- 256 / 512MB DDR2 Onboard
- Support 4S/2USB/VGA/LCD/AUDIO/LAN/GPIO



VDX-6318

Embedded Vortex86DX
Tiny CPU Module

- Vortex86DX CPU 800MHz
- 256 / 512MB DDR2 Onboard
- Support 4S/4USB/VGA/LCD/AUDIO/LAN/2GPIO



VDX-6326

Embedded Vortex86DX
3.5" CPU Module

- Vortex86DX CPU 800MHz
- 256 / 512MB DDR2 Onboard
- Support 4S/3USB/VGA/LCD/LVDS /AUDIO/3LAN/GPIO/CF/FDD



PDX-057T

5.7" Slim and Compact
Design Panel PC

- 5.7" TFT LCD with Resistive Touchscreen
- DM&P Vortex86DX 1GHz with 256MB DDR2
- Compact Flash and Mini-PCI support
- Fanless and low power consumption



Email: info@icoptech.com
www.icoptech.com

ICOP Technology Inc.

Los Angeles Office
Tel: 1-626-444-6666

East Coast Office
Tel: 1-856-546-6555



No Free Lunch with Embedded Operating Systems

Total Development Costs Drive OS Decision

By Cheryl Coupé

The interview is by phone, but the rueful shake of the head is nearly audible as Jerry Krasner, founder and principal analyst for Embedded Market Forecasters says, "I still get calls from CFOs saying, 'You know, we thought Linux was free. We're eight months out and we've run out of our budget – what happened?'" Krasner's research approach is to look at the total costs relating to the embedded operating system choice, including number of developers per project, the percentage of projects that are cancelled, come in on time or on budget, and how close the final design met design expectations. What he has found is that every commercial Linux distribution (including Android) out-performs free Linux in terms of development-cost return on investment (ROI).

And while Linux in all its iterations is firmly entrenched in the embedded world, Android is just beginning to make a splash. Last year, Krasner's research indicated that eight percent of responding developers were using Android, but that response jumped to 18 percent this year. After watching growth of other so-called "can't miss" operating systems such as Symbian, however, Krasner is waiting to see a pattern of year-over-year growth before declaring Android a winner. In the meantime, he is seeing positive trends with Android compared to a range of commercial Linux distributions.

From a development-time and manpower standpoint, Android is competitive with an average of 15 months and 7.4 developers per project. That compares to an average development time of 13.8 months for Nucleus, 18 for Integrity, 16 for VxWorks and 13.7 for LynxOS, while the number of developers per project ranges from 8.1 for Nucleus to between 13 and 15 for Integrity, VxWorks and LynxOS. Android is also compelling in terms of design cost. Embedded Market Forecast sees an industry average for an embedded design of about \$2.3 million, while the average cost of an Android design is \$1.23 million per project. In comparison, average design costs for Nucleus came in at \$1.21 million, Integrity \$2.96 million, VxWorks \$2.6 million and LynxOS \$1.8 million per project.

However, Krasner also found that development strategies are changing and the operating system is the least significant element in terms of containing cost and getting to market. Krasner explains, "Back in 2004 we asked, 'How close to your pre-design expectation was your final design for performance, system functionality, features and schedule?' A full 30 percent of respondents were saying they weren't within 50 percent. That's outrageous. It changed in 2005 and has been improving every year since then. When we worked back through our data

... what we came to find out was that this was the emergence of OS-agnostic IDEs." Krasner notes that in 2005, only seven percent of developers were using Eclipse, compared to 40 percent today.

As developers move towards environments that best suit their needs, he is also seeing a growth in the use of UML modeling simulation, and particularly Rhapsody and Simulink, which allow object-oriented and C developers to work on a project simultaneously. As an additional advantage, if the hardware changes the developer can use the modeling tool to port the APIs already developed to the new hardware configuration. According to Krasner, depending on the vertical, there's anywhere from a 30 percent to 90 percent ROI advantage by using modeling. Embedded Market Forecasters also compared data on companies whose designs were behind schedule and found that companies using modeling were seven times more likely to be in the group that had 30 percent or less of all of their designs behind schedule.

The other trend that Krasner highlights is the move to selecting an operating system based on its appropriateness for the application. He believes that companies are becoming less likely to take on the overhead of a powerful or mission-critical (real-time) operating system if they don't need it. Large pending cut-backs in defense spending are expected to ripple down through suppliers, especially those focused on mission-critical operating systems. In contrast, small real-time operating systems such as ThreadX and Micrium can be used in mission-critical and non-mission-critical applications; ThreadX is now on a billion products, compared to MontaVista on 700 million and Nucleus on 500 million. According to Embedded Market Forecasters, the mission-critical market is only about eight percent of the total in terms of users. That may be large in terms of revenue, but as developers question whether a specific application needs the overhead, there may be a transition to smaller, more manageable operating systems.

Cheryl Berglund Coupé is editor of EECatalog.com. Her articles have appeared in EE Times, Electronic Business, Microsoft Embedded Review and Windows Developer's Journal and she has developed presentations for the Embedded Systems Conference and ICSPAT. She has held a variety of production, technical marketing and writing positions within technology companies and agencies in the Northwest.



Android Dives into Embedded Despite Fragmentation Risks

Linux Takes on Security and New Market Opportunities

By Cheryl Coupé

In a July 2011 research note, VDC Research analyst Jared Weiner stated that the commercial market for Linux-related software and services passed \$140 million in 2010 and is expected to maintain double-digit growth through 2013. Weiner also expects Android-related services and support will be among the largest drivers of growth in the broader Linux market as interest grows among embedded developers. In our roundtable discussion, participants see fragmentation risks running alongside the opportunities for Android in embedded applications. But there's no doubt that Linux in its many iterations is a hot topic. EE Catalog talked to Dan Noal, senior solutions architect, and Nithya Ruff, senior director of product marketing for Linux, both at Wind River, along with Brian Gildon, director of strategic alliances at Enea, and Rod Crawford, principal engineer and Philippe Robin, Linux program manager, for ARM to get their insight.

EE Catalog: How are embedded Linux developers balancing application requirements such as performance, battery life and extensibility for future needs?



Brian Gildon, Enea: Requirements are often application-specific and as we all know subject to change. The key to balancing requirements while developing a future-proof platform is to have an extremely flexible Linux build environment that allows you evaluate multiple configurations – it's really about smart OS configuration. With the emergence of multicore this is becoming even more important with the ability to further balance performance versus power through core management. The emerging platform-builder development model gives developers just such an environment. The platform-builder paradigm permits the building of multiple "custom" distributions by the platform development team, permitting the configuration, build and test of custom Linux platforms derived from multiple sources and tailored to embedded silicon vendor optimizations. In the end, this methodology not only produces the best software for a given set of requirements but is also much more cost-effective and faster to develop.



Dan Noal, Wind River: Each of these requirements is quite hardware dependent and different hardware platforms have different strengths, with power management happening at both the silicon and software levels. A commercial embedded Linux solution will be optimized for all these features, and provides developers with a choice of profiles to meet their requirements.

The popular Linux-based application platforms (e.g., Android, iOS) are focusing on providing platform features that permit the application developer to make some tradeoffs between performance and power consumption. The Android platform provides some basic heuristics to the user of the device to note which applications are consuming the most power. As for extensibility, the appstore / marketplace concept is bringing the notion of extensibility to applications, while the relatively frequent platform revisions from platform owners (e.g., Google, Apple) are showing that a consumer can expect several revisions of the base platform during the life of their device. There is pressure on OEMs to accommodate the platform revisions for a reasonable period after a device is launched.



Philippe Robin, ARM: This is a broad question. The answer really depends on the type of device and environment the developers are targeting. Linux developers use existing power-management features such as CPU-freq, CPUIdle and tools such as Powertop and PowerDebug to optimize how power is managed on given platforms and optimize policies accordingly. Linaro for instance now has a dedicated group working on optimizing this for ARM platforms (<https://wiki.linaro.org/WorkingGroups/PowerManagement/>) and contributing changes and optimizations to upstream projects. ARM and partners are constantly innovating new functionality to further optimize CPUs and platforms for power usage, pushing software developers to take advantage of these to enable products with longer battery life and improved experience.

EE Catalog: Android has lots of momentum behind it in the consumer arena. To succeed in the embedded space, are there specific challenges it must overcome or new opportunities on the horizon that will push it further?

Gildon, Enea: Clearly fragmentation is a huge issue facing Android. Multi-sided software platforms, like Android, that bring together device makers, application developers, content suppliers and consumers require stable, reliable and interoperable software on which to build businesses and, of course enjoy a first-class user experience. The most successful software platforms are the ones that provide a seamless and reliable user experience – no guessing about whether an application will work on not on a given device. As Android expands into devices beyond phones the user experience issue will become even more important.

Dan Noal, Wind River: For each new segment, some specific features are required for broad acceptance. Embedded is really a catch-all phrase that encompasses many segments. Android is a bit of a hybrid – a Linux-based platform with some proprietary licenses involved that may create some resistance among open-source developers. For Android to gain momentum into traditional embedded areas, the embedded community will need to expand the Android platform to suit, which can lead to fragmentation problems. Security is also an issue, as Android is an open platform potentially vulnerable to different types of exploits. We see Google being very active and responsive to reported issues. My feeling is that Android has a good chance in traditional embedded segments that share a lot of the same requirements as the consumer (smartphone, tablet) segments.

Robin, ARM: Android is a very popular platform and is in a growing number of segments and devices. Its popularity and the number of applications available make it a very appealing solution. There are a large number of embedded devices, which are not necessarily as open as smartphones, for which Android may provide more than is actually needed. In such cases, custom-built Linux or RTOS-based solutions may still be more appropriate, due to either specific real-time constraints or the memory footprint available on the device. However, new hardware platforms being developed have ever-increasing capabilities and enable the use of Linux and Android solutions in some of these segments.



Rod Crawford, ARM: I think one challenge relates to the purpose of Android and the primary driving force behind it: Android is a platform for consumer devices, driven by Google. Ultimately, what this means is the roadmap of the platform is driven primarily by Google for consumer devices. The platform is open source and developers are free to tailor it and re-purpose it for, say, embedded devices. However, it is unlikely that

the work undertaken to change or supplement Android for embedded devices will make it in the main Android code-base. Therefore, when Google releases a new version of Android into the open-source world, the work done to tailor and repurpose Android for embedded devices will need to be repeated again on the new version of Android. This may be trivial or it may be significant – there are no guarantees.

EE Catalog: What are some of the key issues that are arising around security in embedded Linux?

Gildon, Enea: As Linux grows in popularity (Android or otherwise), it increasingly becomes a target for hackers and malware. Kernel-level security is quite good as Linux is open source and smart distribution selection ensures that applications cannot take advantage of kernel space easily. User-space security is more difficult, and the primary target of malware in Android. Security for user data and apps will improve as cryptography becomes more user-friendly and transparent. More policing of approved applications is needed by Google. There are a number of movements out there to increase security – (taintdroid and guardianproject.info) are a couple.



Nithya Ruff, Wind River: Interestingly, government agencies like the Department of Defense are beginning to understand that the Linux developer community itself is a security feature—the more eyes on the code, the more likely exploitable bugs are to be caught. And until Wind River created and certified an EAL 4+ embedded Linux platform, many people said it couldn't be done. One of the challenges has been the fact that SE Linux, which is a good solution for hardening Linux, was difficult to configure. If it wasn't applied right, it couldn't do the job. Some of the most significant security issues arise from the sheer number of connected devices in the market today, and the fact that until recently, device security had been a relatively ad hoc affair.

Robin, ARM: Security needs to be approached with a holistic system view and not limited to a specific OS or individual technology. Much of this is governed by what the hardware platform can offer to enable security. In the case of ARM platforms, solutions like as ARM™® TrustZone™ can form part of the solution when closely integrated with software.

EE Catalog: What are some of the most exciting trends or application areas you're watching in Android/Linux development?

Gildon, Enea: The rise of the multi-OS device that blends Linux, real-time operating systems and in many cases real-time executive environments is a trend that is taking off in high-performance market segments. Developers are seeing the

advantages of combining the benefits of the Linux software and tools ecosystem with the power and reliability of real-time environment. The rise of multicore processors with 8, 16 and even more cores has made the multi-OS device a reality but developers continue to struggle with device debugging and optimization given the vast amount of configurability and complexity at their fingertips. At Enea, we think the next really important development is the tooling needed to maximize performance in multicore, multi-OS development.

Nithya Ruff, Wind River: I see a kind of cross-breeding taking place – for example, Android moving beyond mobile phones and tablets and embedded Linux showing increased real-time and virtualization capabilities. Both platforms are moving into new industries. Both are becoming more secure and capable of more sophisticated graphics and multimedia. One of the most exciting trends I see in the open-source community is collaboration across the commercial and open-source embedded community around tools and infrastructure, as you find in the Yocto project, an open-source project that provides templates, tools and methods to help create custom Linux-based systems for embedded products regardless of the hardware architecture. This will simplify and streamline the creation of Linux distributions for embedded devices.

Robin, ARM: With the recent Android Open Accessory support initiative, it will be interesting to see how this develops

and enables interaction with a new range of devices and use cases. Android is increasingly being used as a software platform for new segments, so it will be interesting to see how this translates in terms of new applications and use cases.

Crawford, ARM: I think one of the most interesting trends now is the “Internet of things.” We have been able to connect sensors and actuators to the Internet for some time, but really, there has been a lack of cohesive framework for this activity, as well as lack of thinking big. With Android@home, Google is making Android the central monitoring and control hub for the Internet of things within your home and beyond... your street, your city, your planet. The Internet of things when seen at this scale, with Android as the enabler to connect the things to the cloud, means our devices will become far more intelligent and react not only to their own ambience but also to those around the world.

Cheryl Berglund Coupé is editor of EECatalog.com. Her articles have appeared in EE Times, Electronic Business, Microsoft Embedded Review and Windows Developer's Journal and she has developed presentations for the Embedded Systems Conference and ICSPAT. She has held a variety of production, technical marketing and writing positions within technology companies and agencies in the Northwest.



Embedded Linux and Android **ONLINE** www.eecatalog.com/embeddedlinux



Explore...

- Directory of leading Embedded Linux and Android Solutions
- Top Stories and News
- White Papers
- Expert Opinions (Blogs)
- Exclusive Videos
- Valuable Articles
- Ask the Experts

Sign up for the quarterly Embedded Linux and Android E-Product Alert

The Only Conference Focused on AdvancedTCA, AMC, and MicroTCA!



The Right Solution for High-Bandwidth Systems

AdvancedTCA Summit is the only event with all the major vendors, industry associations and key people working on AdvancedTCA, AMC and MicroTCA. These PICMG specs handle the latest processors, newest interfaces, and most demanding high-availability applications. This is the one-stop shop for evaluating AdvancedTCA and MicroTCA products and designs in telecom, storage server, embedded systems, medical equipment, instrumentation businesses and military/defense/aerospace systems.

- ***Learn to develop next-generation networks and wireless systems.***
- ***Identify major market trends.***
- ***Meet with potential partners, distributors, consultants, and solution providers.***
- ***See the latest products!***



November 1-2, 2011

AdvancedTCA Summit & Exhibition

DoubleTree by Hilton Hotel San Jose
2050 Gateway Place, San Jose, California USA



REGISTER ONLINE

www.advancedTCAsummit.com



Yocto Project – Big in Embedded Linux

By Rudolf Streif, director of embedded solutions, The Linux Foundation

Yocto, the prefix in the metric system denoting a factor of 10⁻²⁴, gives the name to an open-source collaboration project that greatly streamlines the creation of a custom Linux distribution for embedded devices. However, saying that the name is an understatement is an understatement itself.

The Yocto Project (www.yoctoproject.org) is a complete embedded Linux system development environment with tools, recipes and documentation. The Yocto toolbox is self-contained and includes compilers, debuggers, packagers, an emulation environment, an application toolkit generator and many more, making it the ideal ready-to-go solution for embedded Linux developers. And yet, it is easy to use. The available resources, templates and documentation cater to both novice and experienced embedded Linux developers.

Community-tested images covering various build profiles across multiple architectures including ARM, MIPS, PPC, 32- and 64-bit x86 provide a solid starting point for new engineering efforts. Specific hardware platforms are supported through board support package (BSP) layers for which the Yocto Project has developed a standard format.

The icing on the cake is the Application Toolkit Generator that automatically creates a complete environment for engineers to develop applications on top of a platform created by the Yocto Project. This environment seamlessly integrates with a plug-in for the popular Eclipse IDE and can be shipped by device vendors together with their hardware platform to jump-start application developers.

Why Linux? Why Yocto?

A recent survey of embedded developers conducted by Embedded Market Forecasters (www.embeddedforecast.com) reported that two primary factors contributing to embedded developers' choice of operating systems are cost (44.6%) and the availability of source code (33.1%). From this point of view, Linux seems to fit the bill quite nicely. However, Linux puts up some hurdles that not too long ago were hard for embedded systems and their developers to jump.

Linux requires a file system. For Linux to run it must have a file system with read and write access. Until the ubiquitous availability of inexpensive and long-term reliable

flash memory devices that typically meant a hard drive, which is not practical for most embedded use cases.

Linux requires a memory management unit (MMU). Linux is a multi-tasking operating system; effective task-switching mandates that individual processes have their private space in physical memory that easily can be mapped into the CPU's address space. Typical microcontrollers that are widely used for embedded applications do not provide an MMU. That did not pose too much of a problem in the past since embedded applications were mostly single-tasking and context switching was limited to interrupt service routines (ISRs). However, today's systems-on-chips (SOCs) provide multi-core CPUs with MMUs to meet customers' demands for connected devices. No consumer would accept anymore that he or she cannot access the Internet or have a look at the calendar while conducting a phone call on a smartphone.

Linux is non-monolithic and multi-sourced. While this is essentially a good thing because it lets developers easily customize the OS to fit their particular embedded devices, it poses the question of where to get a Linux distribution that fits the needs of embedded devices. For desktops and servers this problem has been solved by the many readily available distributions, but for embedded devices there is no such thing as a plug-and-play Linux distribution. And that is simply due to the fact that hardware for embedded devices is naturally much more diverse than standard desktop and server computers. That is where the Yocto Project provides a solution.

What is Yocto?

Contrary to popular belief, Yocto is not an embedded Linux distribution, but a kit of tools and recipes to create a custom Linux distribution from source. It relieves the embedded developer from identifying the proper packages that make up a Linux distribution and download their sources as well as from setting up a system with the correct set of tools to build these packages and finally assemble them into an image that can be booted on the target device.

The Yocto Project builds upon and extends a series of upstream open-source projects which form its toolkit. With each Yocto release, updated versions of these projects with new features and bug fixes are included. Every

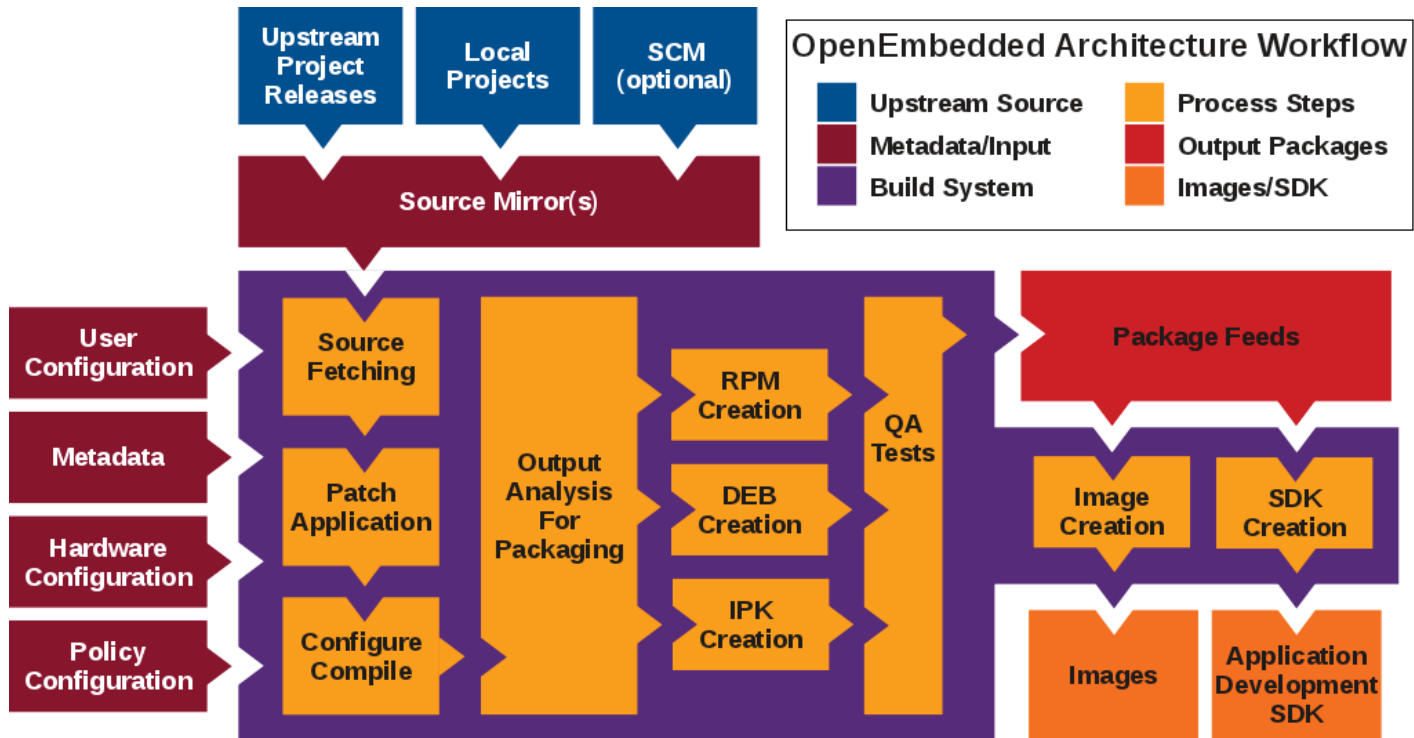


Figure 1: OpenEmbedded Architecture Workflow (Source: www.yoctoproject.org)

new release of Yocto also provides updated recipes and templates to track changes, features and bug fixes of the source packages that make up a Linux distribution such as the Linux kernel.

At the center of Yocto is Poky (www.pokylinux.org), a platform-independent, cross-compiling layer that utilizes the OpenEmbedded core (www.openembedded.org). Poky provides the mechanism to download, patch, build and combine thousands of distributed open-source project to form a complete and coherent but fully customizable Linux software stack.

Figure 1 depicts the Yocto workflow based on OpenEmbedded. User Configuration, Metadata, Hardware Configuration, and Policy Management determine what Upstream Projects, Local Projects (stored on the developer's system) and what sources from optional Source Change Management systems to include, as well as control the individual process steps of the workflow. A Yocto Project release already contains configuration information and metadata that will produce a working Linux system image only requiring very minimal adaption to the developer's local build environment.

While Figure 1 shows a rather linear process consisting of the steps Source Fetching, Patch Application, Configure/Compile, Output Analysis for Packaging, Package Creation and QA Tests, these steps are in fact repeated for each source package and therefore potentially many thousand times before all PackageFeeds have been created and can

be combined into an image. Therefore Yocto supports multi-processor and multi-core build systems by automatically calculating dependencies and executing process steps in parallel, hence greatly accelerating the entire build process. Of course, Yocto also manages changes and only rebuilds Package Feeds whose input, source, metadata, dependencies, etc. have changed.

To further simplify the process for novice as well as experienced users, Yocto provides a series of different profiles that automatically create images for common applications:

- **poky-image-minimal** – A small image just capable of allowing a device to boot.
- **poky-image-base** – A console-only image that fully supports the target device hardware.
- **poky-image-core** – An X11 image with simple applications such as terminal, editor and file manager.
- **poky-image-sato** – An X11 image with example GUI implementation using Matchbox Window Manager, Sato theme and Pimlico applications. The image also contains terminal, editor and file manager.
- **poky-image-sato-dev** – An X11 image similar to poky-image-sato but also includes a native toolchain and libraries needed to build applications on the device itself. The image also includes testing and profiling tools as well as debug symbols.

- **poky-image-lsb** – An image suitable for implementations that need to conform to Linux Standard Base (LSB).
- **meta-toolchain** – This profile creates an archive that contains a standalone toolchain that can be used externally to Yocto. The archive also contains QEMU and the scripts necessary to run Yocto QEMU images.
- **meta-toolchain-sdk** – This profile creates an archive that contains everything of the meta-toolchain profile but also includes headers and libraries to form a complete standalone SDK.

The above list represents the most commonly used profiles. Many more specialized profiles are available and the list continues to grow.

Figure 2 shows the example GUI using Matchbox Window Manager with Sato theme desktop running from an image created with Yocto using the core-image-sato profile.



Figure 2: Yocto image created with core-image-sato

Who Supports the Yocto Project?

The Yocto Project is a Linux Foundation Workgroup. The project was inceptioned by Richard Purdie who has been appointed as a fellow of the Linux Foundation. In this capacity he is working full-time on the Yocto Project, OpenEmbedded and Poky.

The Yocto Project software is designed and developed through a collaborative effort by an open community of professionals and volunteers and is supported by a series of participating organizations. It is open for collaboration to anyone who is positively contributing. Like many other open-source projects, the Yocto Project employs a governance model consisting of various groups:

- **Contributors** – Professionals and volunteers contributing to the collaborative effort: code developers, documenters, users, etc.
- **Maintainers** – The delegated authorities who review patches that are submitted by contributors and technical leaders, provide feedback to the developers and accept the ones that are appropriate.
- **Technical Leaders** – Along with maintainers, the technical leaders ensure excellence and functionality.
- **Interest Groups** – These groups define and document the various requirements for the Yocto Project, which are formed around the strongest needs from the users.
- **Architect** – The Project's architect provides overall technical leadership and resolves conflicts.
- **Advisory Board** – The Project's advisory board influences the interest groups to ensure that requirements are set for the Project. Its range of responsibilities also includes creating and resolving working groups to support the fulfillment of the project objectives, appointing maintainers and technical leaders, and defining policies and procedures.

Corporate supporters of the Yocto Project include semiconductor companies, operating system vendors, OEMs and others. Through collaboration in the project they effectively are aligning their efforts to everybody's benefits. Semiconductor companies are lowering their R&D cost for providing software support for their SoCs; OSVs shorten their product development life cycles through reduced customization effort; and OEMs can focus on differentiating their products rather than developing basic and common functionality. Companies contributing to the project today include Cavium Networks, Dell, Freescale Semiconductor, Intel, LSI, Mentor Graphics, Mindspeed, MontaVista Software, NetLogic Microsystems, RidgeRun, Texas Instruments, Tilera, Timesys and Wind River, among others.

Rudi Streif manages The Linux Foundation's initiatives for embedded solutions, working with the community to provide environments and platforms for embedded Linux systems. Mr. Streif has an extensive background in embedded software development and bringing products based on Linux to market.





RTECC

Real-Time & Embedded
Computing Conference

TRANSPORTING ENGINEERS

INTO THE WORLD OF EMBEDDED

ARE YOU BEING LEFT BEHIND?

- Market-revealing **keynote speakers**
- Technically focused embedded **seminars and workshops**
- Vendors demonstrating **newest technologies**
- Network with the **brightest engineers**

Can you afford to miss it? **FREE** admission, lunch, parking and prize drawing entries at each event. RTECC is your best opportunity to discover a new world of possibilities within the embedded market.



Scan this QR code with your smartphone and **REGISTER TO ANY UPCOMING RTECC EVENT INSTANTLY!**

Go to RTECC.com to download a QR reader

UPCOMING LOCATIONS

Denver, CO
08/09/11

Salt Lake City, UT
08/11/11

Irvine, CA
08/23/11

San Diego, CA
08/25/11

www.RTECC.com
REGISTER FOR FREE TODAY!

From Zero to Boot: Porting Android to your ARM platform

By Vassilis Laganakos, senior embedded software engineer, ARM Ltd.

If you are in the embedded world, mobile phone industry or just a gadget-loving person, Android is a word you hear every day. This article gives an overview of the procedure for getting the popular OS, primarily designed for ARM®, running on your favourite ARM-based system-on-chip (SoC) platform. In this article, we point out potential pitfalls, challenges and other issues that you may encounter.

Terminology

The following terms are used:

- Mainline kernel – the Linux kernel that you can get from mainline <http://www.kernel.org>
- Reference kernel– the kernel you use for your board; most likely a Mainline kernel, plus patches required to support your hardware.
- Merged kernel – the result of merging the Android kernel with the Reference kernel.
- Android kernel– the kernel available from Google’s Android Open Source Project. It contains Google’s additions to the Mainline kernel. See <http://android.git.kernel.org>

Zero

Android’s software stack has been developed and tested on many ARM platforms, so there are only a few things you need to change to get it to work on another ARM platform.

The procedure to port Android is composed of two parts: the kernel and the Android userspace filesystem. For the first part, find and merge the differences between the Reference kernel and a matching version of the Android kernel, and configure it for Android.

For the second part, things are simpler: just add your platform to the Android build system and build. It should just work. Google provides instructions on how to setup a build environment (<http://source.android.com/source/initializing.html>). Configuring the Android filesystem to fit your boot media and board set-up is important but not complicated.

The most common mistakes made are during the configuration step. Make sure to complete this step for both kernel and filesystem.

Kernel Side

The general rule for the kernel is, “if your Reference kernel does not work on your platform, then the merged Android kernel won’t work either.” It is therefore vital that your Reference kernel is working correctly on your platform before you begin to merge the Android kernel.

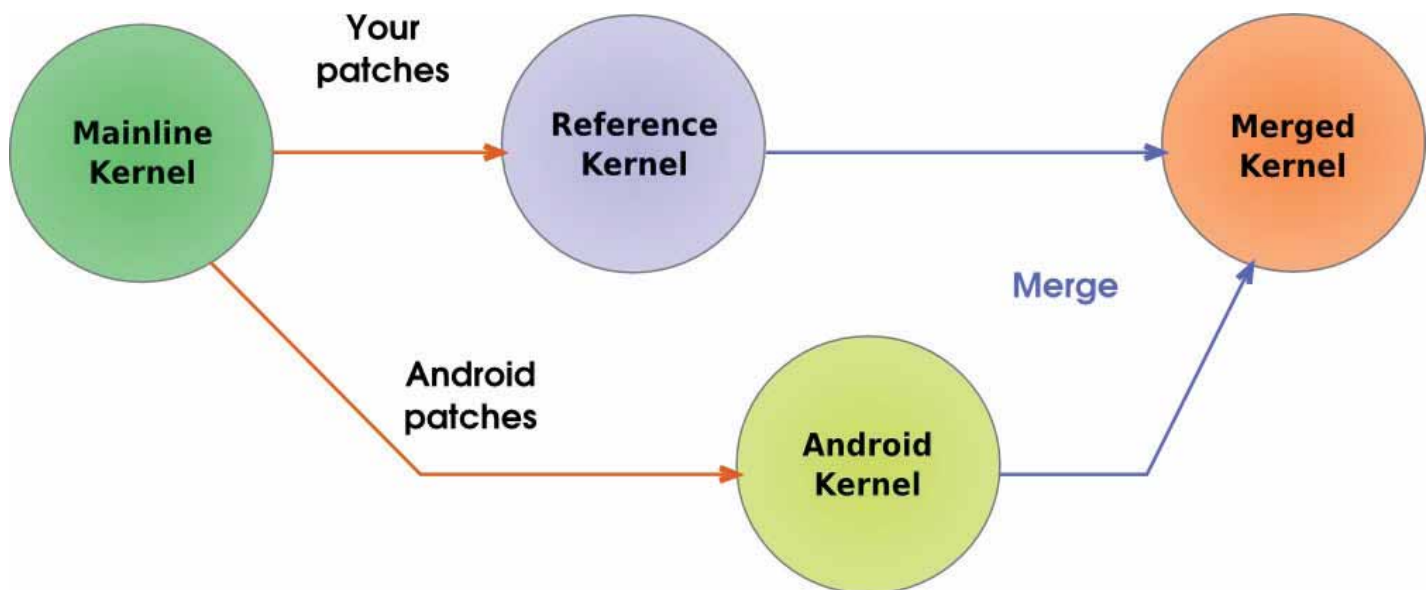


Diagram 1: Kernel Merging Procedure

The bulk of the work here is to produce a merged kernel source tree that contains both the changes to your reference kernel and the additions to the Android kernel.

Working Linux Reference Kernel

Test to make sure you have a working Reference kernel and a Linux root filesystem functional on your board, i.e., boot to a Linux graphical environment. The kernel should include all of the appropriate support required for a windowing system, such as graphics drivers for the framebuffer or your graphics chip. You should also make sure that these drivers support the desired screen resolution. The Linux root filesystem is only required to ensure the Reference kernel works and to test the Merged Kernel.

It should be noted here that the Android releases after Cupcake require double buffering and page-flipping support from the graphics system, so you need to have support for this in your Reference kernel.

Since the correct configuration of the kernel plays a major part, keep a copy of the .config file of the functional Reference kernel, as you will be using it as a base configuration for the Merged kernel.

Misconfiguring the kernel can cause the Android boot to fail on your device.

Android Kernel

The Android kernel should have the same revision as your Reference kernel. If that is not possible, use the closest kernel available. A usual trap for developers is use the latest Android kernel because it contains all the new features advertised. Revision mismatch between the two kernels will result in spending more time merging and debugging.

Merge the Reference with the Android kernel

The recommended way to merge is to use git with the common ancestor of the two trees. The “git-merge” command will produce a single tree that will keep the patching history and maintain all individual commits for you to search and investigate. This is indispensable when it comes to merging and debugging.

Configure and Build the Merged Kernel

Use the configuration file for your Reference kernel, saved earlier, to build the Merged kernel. During the configuration step, you may be asked about options that were not available in your Reference kernel due to the Android parts in the Merged kernel.

Once built, confirm that the Merged Kernel boots and runs with your working Linux root filesystem.

If it does not boot, take a step back and make sure merged code from the Android kernel source tree did not break anything

in your Reference kernel. Start with the files in which you resolved merge conflicts.

Since this step is board-specific, there is no silver bullet to make things work. The best practice is to use the same revision kernels.

If it does boot, you can then add the options required for Android. This is the most important step of the port. You will at least need to have power management, wakelocks, ashmem, pmem, switch class support, android binder ipc, android logger, android adb and pda power supply enabled and working in the kernel. Be aware that configuration variable names tend to change between kernel versions.

In multiprocessor systems, CPU hotplugging support is required. You should enable this if your ARM-based SoC has more than one core and you wish to boot a kernel with multiprocessor support.

When it comes to building the kernel, make sure that you use a toolchain version proven to work for that purpose.

Userspace - The Android Filesystem

To test whether the Merged kernel works and to complete the port, the Android filesystem is required. This is straightforward and in the worst case, you can just add and build a generic one for ARMv5 architecture, which will not give you the performance but will give you a starting point.

Add and Configure your Device in the Android Source Tree

Each build target defines the configuration for a platform and selects which sources should be built for Android. Google used to provide detailed instructions for adding support for a new device, but this information is no longer available. Nevertheless, some basic guidelines are available on the Internet.

Android's parts are mostly written in portable C code, apart from some ARM-assembly code – mainly for the JIT engines, Dalvik and V8. Since the built binaries are ARM, the ARM ABI allows them to run in any compliant architecture version. For example, an ARMv5 binary will run on ARMv7 architecture platform. In Froyo, the support for two ABIs allows the distribution to run native ARM applications built for one specific architecture version (made with Android NDK).

If you wish to add a non-supported ARM architecture version (e.g., ARMv6k), you need to provide to Dalvik the appropriate files that will allow it to generate code for that architecture version. This is required if you are targeting performance, but you should not do this in the initial port.

Build Android for your SoC

Building the generic Android filesystem is simple: use make. You will need to include additional arguments if you have defined your own platform and configuration. Using the tools

and scripts provided from the Android source tree will give you reliable result and better control over building specific parts. Note: you do not need to build Android as the root user.

Configure the Android Root Filesystem

Once built, boot the Android root filesystem from USB sticks, MMC/SD cards or an NFS Server. There are a few things to check:

- Make sure the kernel has support for the media you are going to use.
- Ensure you have copied the contents of the “system” directory to the empty “system” directory inside “root.”
- Be certain ownership and permissions of the files have not been modified after copying them from the build system.
- In the configuration file (init.rc):
 - Comment out the lines, which remount the filesystem as read only.
 - If you need root privileges in the shell, set the “service console” user to “root.”
 - Add your network’s DNS server with “setprop net.dns1 [DNS IP].”

Boot

Your Merged kernel should now be ready to run Android. Load the kernel on your board, set the appropriate boot arguments and Android should boot shortly.

The first-time boot will take longer since Android is optimizing the .dex files (using dexopt), so be patient. Use the dex-preopt tool on .jar files before you install them to avoid this delay.

Should any minor issues appear, experiment with the kernel options and the init.rc configuration, based on the information you get from Android’s logcat and the system’s dmesg tool.

Logcat and dmesg are your Swiss-army knives. Use them to identify and investigate any possible issues.

Post-Boot

You should now have a working Android OS on your ARM-based SoC.

Although Android is based on the Linux Kernel, it is not using the GNU/Linux filesystem. Therefore, it is not Linux. Expecting Android to work like or provide the same functionality as traditional GNU/Linux systems is a mistake that most of us have made!

For example, the C-library (Bionic) is different, and you will not get the standard dynamic linker. Therefore, if you want to run pre-built GNU/Linux binaries, they have to be statically linked.

The shell is a bit restrictive, with not as many tools usually available in GNU/Linux, but you could drop in a statically linked version of BusyBox that provides the ones you are missing.

And don’t forget that adb is your friend!

Summary

Porting Android to new ARM-based SoC should be easy and straightforward, if you have a Linux kernel port for your SoC. From experience, merging a Reference kernel and the Android patches of the same revision requires about a day; however in the case of revision mismatch, more time may be required. The Android filesystem part of the story may work without changes, and if Dalvik already supports the ARM architecture version that your platform’s SoC is using (e.g., ARMv7-A), you will also see improvements in performance over a more generic build.

For more technical information on the subject, please visit: <http://blogs.arm.com/software-enablement/498-from-zero-to-boot-porting-android-to-your-arm-platform/>

Vassilis is an undercover physicist in the computing industry. He got his MSc in advanced computer science in Manchester, UK, where he focused on high-performance computing, micro-kernel operation and principles, and got addicted to embedded systems through the ARM architecture and the SpiNNaker project. He is very curious about almost every tech subject in the field, old-timer in Linux and FreeBSD, and loves ++finding better ways to code “that thing”!



How Do You Eat a Multicore Elephant?

By Stephen Olsen, Mentor Graphics

The availability of multiple symmetric CPUs on system-on-chip (SoC) has increased the appetite of system architects to design with more cores for even greater processing. It's important to keep in mind however, not all software is ready for the main course. Just because you can run many processes and tasks on multicore hardware doesn't mean that it will be the most efficient at processing data.

Remember that a multiple CPU design is not a new idea. A look into history reveals systems with multiple CPUs that share memory and have been in existence for at least 25 years. But many of these systems were loosely coupled and did not have cache. Modern SoCs have multiple symmetric CPUs that are tightly coupled and caching can actually make things worse.

It comes down to the type of multicore elephant you're trying to serve up. For example, how much data will be shared in the processing necessary to run your system?

Just because you can run many processes and tasks on multicore hardware doesn't mean that it will be the most efficient at processing data.

A Typical Example

Let's take a look at one way to divide the shared data processing problem. Normally within a long-distance carrier network, control and data planes are used to route phone conversations. A single control entity, the control plane, processes the thousands of calls responsible for routing and billing. The data plane is responsible for handling individual conversations that need to be processed in a minimum response time. In addition to handling individual phone conversations, the data plane must be processed with consistent minimal delays while the

system processes thousands of simultaneous conversations. The problem with this elephant, if you will, is that it has multiple data sets that are completely unrelated but they are all running on a single set of instructions.

While the control plane needs to process many sets of data into a single or even multiple remote databases for billing

purposes, it may also need to re-route as other network nodes become more or less saturated, or even unresponsive. An operating system such as Linux or Android can run on one set of cores responsible for the control of the

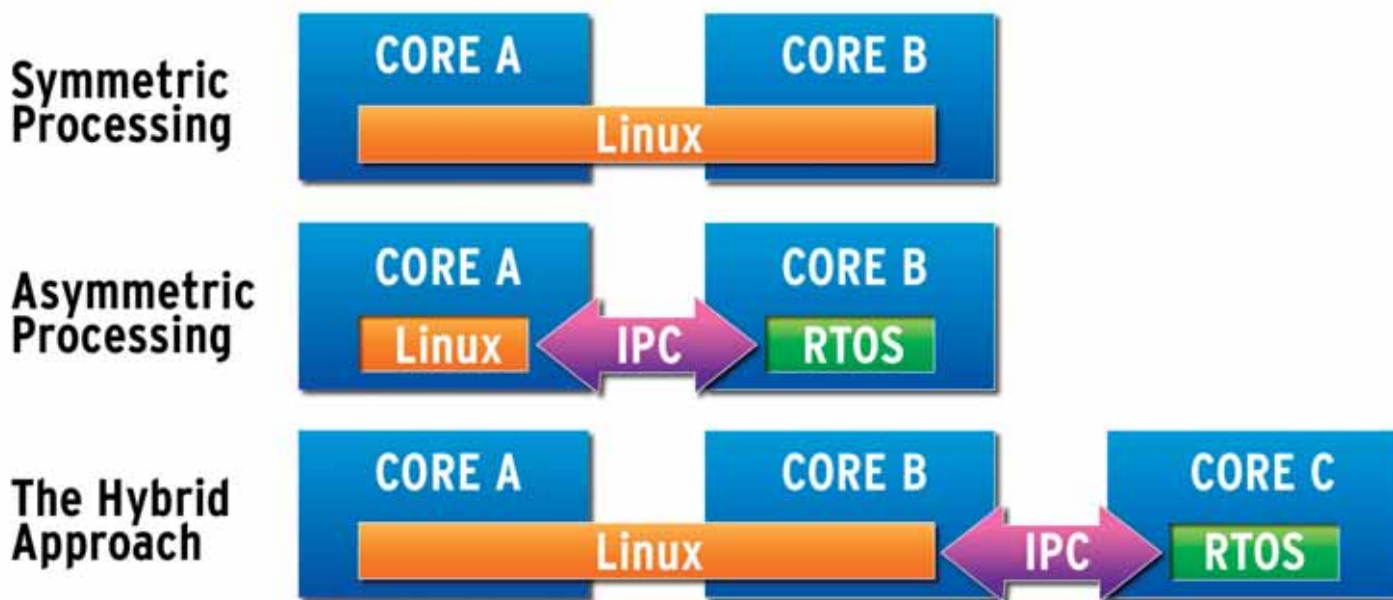


Figure 1: There are a number of approaches to multicore processing.

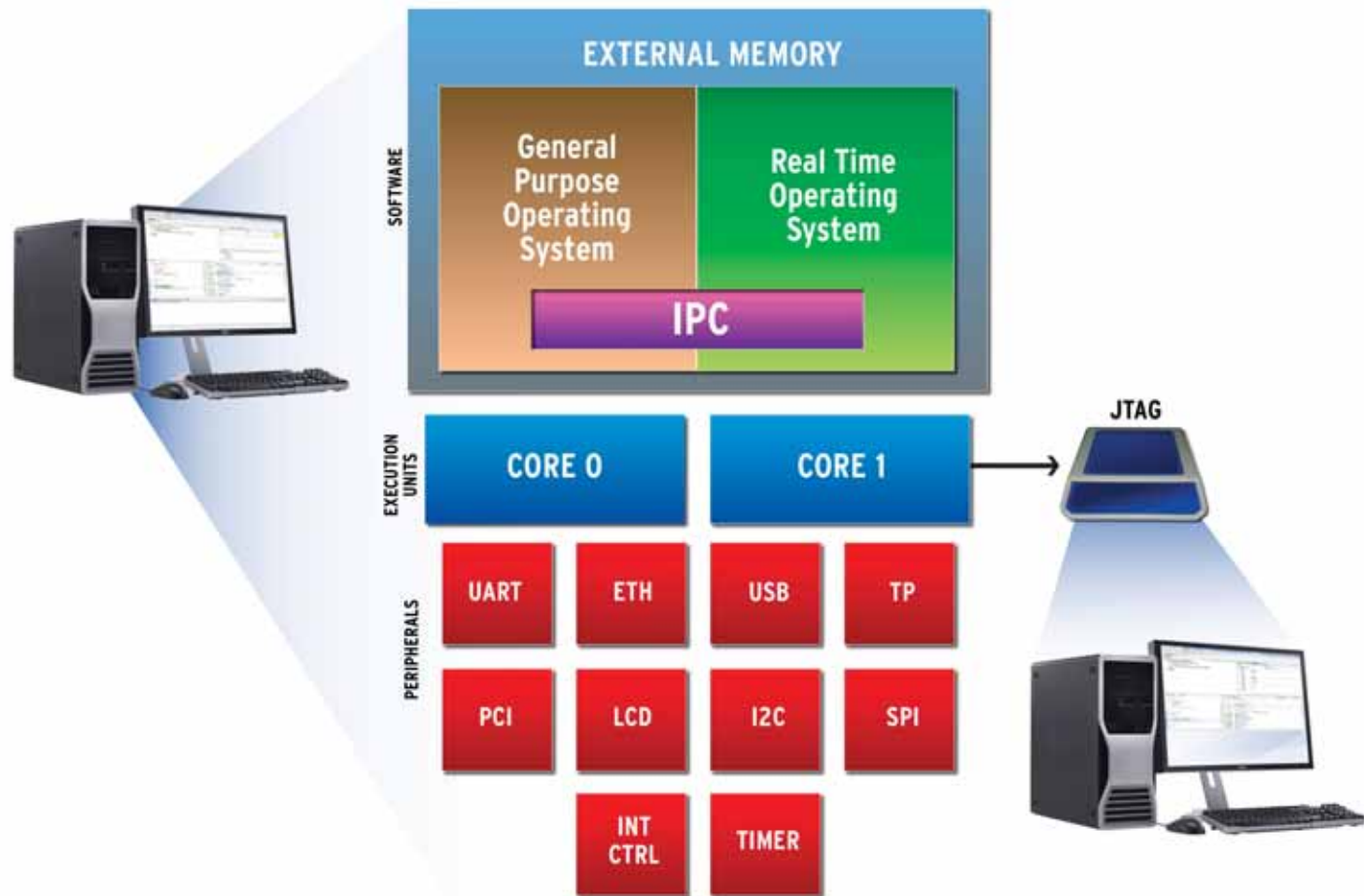


Figure 2: Multicore multi-OS debug scenario with two different connection methods.

network node, while the remaining cores run on a series of real-time operating systems (RTOSes) to process the actual data through the network (Figure 1).

How to Divvy Up the System Resources

For our carrier network example, let's say the system consists of a set of eight symmetric CPUs, 4GB of shared memory and four network connections.

The control plane is issued two CPUs, 2 GB of RAM and one network node. This allows the control plane to keep much of the control database structure in memory as it acts on it. Linux was chosen for the general-purpose operating system because of its available set of middleware necessary for processing control database commands and the limited need for a user interface. We could have just as easily selected Android had there been a user interface requirement or third-party software integration requirement.

The data plane is then divided up into three sets of two CPUs and 512 megabytes of dedicated RAM each. An RTOS was selected for its deterministic behavior, low latency and a minimum of operating system overhead.

The remaining 512 megabytes are available for inter-process communication (IPC) via shared memory between

operating systems. The Multicore Communications API, or MCAP, will be used for communicating between operating system domains.

With the System Partitioned, How do We Know It's Right?

While it's easy to guess, we don't actually know if we made the right allocations until we see where the bottlenecks occur in the system.

Debuggers and profilers are essential tools for multicore design. Whether all eight cores are being used together under a single operating system, or separate operating system domains are used on separate cores, message passing needs to be optimized. Understanding what parts of the system are overloaded and which ones are idle will tell us where to focus for optimizations. The assumption made in the previous example was that each data plane was best served by two cores, but that may result in either overkill or underkill. How many and how large are the messages between cores?

Under Linux there is a tracing tool called LTTng (Linux Trace Toolkit next generation) which can be found at lttng.org. This gives visibility into the Linux side, including down into the IPC mechanism from the Linux operating system domain. For the RTOS, a similar method of tracing

must be available to see what's going on in a running system. These solutions are intrusive and will take both processing and large buffers for trace data. Tracing can be turned off once the system is largely optimized. The real trick is to get all the traces synchronized and to visualize how data flows between operating system domains. This can be handled with a common timebase between all operating system domains using a common viewer.

MCAPI to the Rescue

Having a standard IPC mechanism that can be shared between each node allows for the propagation of tasks in a system in order to balance the systems with minimal porting efforts. The Multicore Association (MCA) created a standard IPC mechanism called the Multicore Application Programming Interface, or MCAPI. With MCAPI, an application becomes portable across both the RTOS and the GPOS if they have similar APIs as well as similar threading mechanisms, like POSIX. Incidentally, Mentor Graphics has released "open MCAPI," which is an open source implementation of the MCAPI specification, comprising a Linux library and kernel driver that enable applications to communicate across cores. The open MCAPI offering was designed to help optimize multicore and multi-OS applications on a single chip.

Between operating system domains we can trace how the MCAPI IPC mechanism is functioning to determine the frequency in which the OSes are interrupted by IPC traffic and how fast they are responding to the request. If one node is overloaded, it can pass some traffic to another node that is mostly idle, thereby balancing across the set of nodes and minimizing the total delay for all conversations. But how quickly can the system balance itself? Can we optimize the point at which a voice conversation is migrated?

One possibility is to allocate all six CPUs to a single RTOS domain. However, all the data sets could migrate between CPUs, causing cache coherency issues. This can cause undue delay in processing a message because the data can be in one CPU cache and then acted upon in a separate CPU. Limiting the number of CPUs that can act on a data set can improve performance. But you'll only know for sure once you put your system through a real-world condition and monitor its behavior (Figure 2). In addition, the next generation product may have 16 or 32 CPUs and this method does not scale.

Tools to Consider

With limited tools available, there are two areas that require special attention: disabling system interrupts and task priority. If there is a common API for disabling system interrupts, it's easy to see where the code is doing this. Each instance can be inspected for what data structures are being protected by the disable interrupt command.

A system object called a semaphore can be used to block multiple entities from accessing the resource at the same time. The operating system must maintain the semaphore structure across all threads that may be simultaneously active. If the operating system is multicore safe, it will take care of the semaphore being accessed with the use of a spinlock that guarantees that system resources like a semaphore are accessed only by one thread at a time. But this does cause a minimal delay on all semaphore accesses.

The assumption that a higher priority thread would be active, thereby eliminating any contention in the system, will need to be inspected. If there are any structure references from different threads, then this contention needs to be eliminated by using a semaphore to guard access. This is a slow process, but tools that profile a running system may be helpful. There may be a race condition in the code that is not easily detected. Code inspection for threads that access the same structures is a necessary step to make sure your code is multicore safe.

In Conclusion

Analysis tools that allow a system architect to see what's going on in the system are key to balancing the system load and optimizing the use of all the system resources. Utilizing the benefits of both a GPOS (Linux or Android) as well as an RTOS can be an advantage by breaking down the problem between control and data planes. Using multiple operating system domains along with a standard IPC mechanism, such as MCAPI, is an advantage that allows threads to be migrated to other operating system domains minimizing any system bottlenecks.

So how do you eat the multicore elephant? One byte at a time. Or with multicore, that would be many bytes at a time depending on how many CPUs you can dedicate to each operating system domain – and how easily it is to break the elephant into several largely independent chunks. Bon Appetite!

Stephen Olsen has over 20 years of embedded software experience with his past 14 years spent at Mentor Graphics. Stephen is currently technical marketing engineer for the Mentor Embedded Software Division. During his tenure at Mentor, he has co-chaired VSIA's Hardware dependent Software (HdS) design working group and authored several papers on system design and power management. Stephen has worked in consulting, system architecture, embedded software, and IP. He holds a BS in physics from Humboldt State University in California.



Utilizing Google's New Open Accessory Framework for Android™ Accessory Designs

By David Flowers, principal applications engineer, Advanced Microcontroller Architecture Division, Microchip Technology Inc.

At the annual Google I/O conference in May 2011, Google announced that they were adding a framework into the Android™ operating system for USB-based accessories. This accessory framework was added to the Android 2.3.4 (API level 10) and Android 3.1 (API level 12) operating-system releases. Google enabled several different options for accessory development. Accessory developers need to understand the options that are available, the design implications of each of these options and how these selections will affect their target market in order to determine what is best for their application.

USB Peripherals and Hosts

Before discussing the features of the accessory framework, it is important to understand a few related concepts about USB. USB is logically a point-to-point communication system between an attached peripheral and the host controller. Peripherals can never talk to each other through USB; there is always exactly one host on the bus. Typically in a USB system the USB host is the more powerful machine (more memory, more CPU power and better connectivity, and it is going to typically be the focal point for the user), while the peripheral is often a simpler device with basic functionality on the bus as a service to the host.

The host also has certain responsibilities on the bus. The first main responsibility is to supply power to attached peripherals. The second main responsibility is to manage the devices on the bus. Because of these differences, the USB host-enabled device and USB peripheral devices often have very different designs.

Android Accessory as a USB Peripheral

As an accessory designer, the choice between these two USB options might seem clear, initially. An accessory to a smartphone/tablet makes the most sense as a USB peripheral. The phone or tablet is likely to be the central focus point for the user, and the accessory is likely going to be providing a service for the phone/tablet. Having the phone/tablet supply power to the accessory is also an attractive option. The option to use the Android phone or tablet as a USB host was enabled in the Android 3.1+ devices. However, this option is not available in other operating system versions.

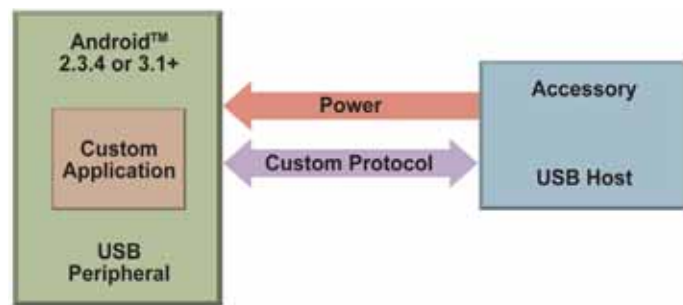


Figure 1a – Creating an Android accessory, where the accessory is a USB peripheral but isn't using a standard USB class.

Android Accessory as a USB Host

When Google released their accessory framework—called the OpenAccessory framework—they recognized that many Android devices already in the marketplace were designed only for USB peripheral capability. These phones and tablets don't have the hardware required to operate as a USB host. In order to enable these products to attach to an accessory, Google had to create an accessory framework option based on the accessory being the USB host and the Android phone/tablet being the USB device. This OpenAccessory framework option is enabled in both the Android 2.3.4 and Android 3.1+ operating-system versions.

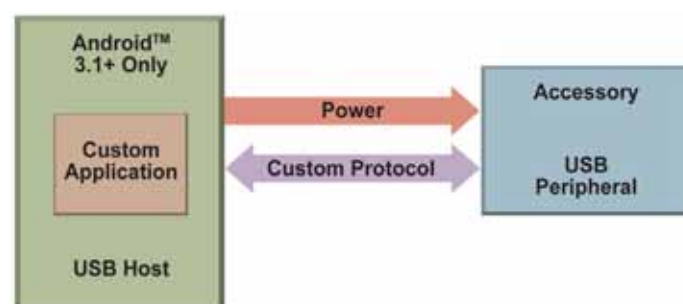


Figure 1b – When creating an accessory that needs to support Android devices that don't already have USB host capability or are running an OS version that doesn't support the USB host mode, then the accessory must be the USB host and use the OpenAccessory framework to talk to the Android device.

Android Accessory for Standard USB Accessories

The third option that is available is native OS support. With the addition of USB host capabilities to the operating system, support for some standard accessories was also added, such as mice, keyboards, thumb drives, etc.

For USB peripheral manufacturers who make these types of standard accessories, they don't need to worry about creating new devices or software to work with Android phones/tablets. Their accessories will work on any operating system that supports that specific standard class driver. All three of these support options are highlighted in Figures 1a, 1b and 1c.

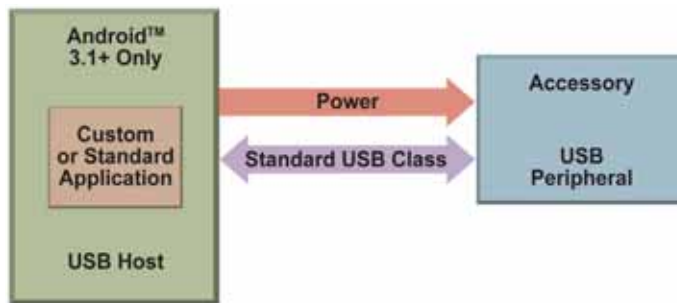


Figure 1c - An Android accessory using a standard USB device class.

Which One Should I Choose?

Selecting one of the three available options for creating an accessory requires a little information about the implications involved with each. The first major question that needs to be answered is the target set of Android devices for the accessory. In order to create an accessory that is a USB device, it requires that the target Android phone/tablet have the built-in hardware capability of being a USB host. Even if the hardware is capable of running the USB host, unless the Android device is running Android 3.1+, it will not be able to use either of the USB host options. As of June 1, 2011, Android 3.1+ was installed on 0.3% of Android phones/tablets. While the percentage is low right now for Android 3.1, an accessory designer needs to weigh the options of cost/features against what the Android 3.1+ adoption rate will be when the accessory is released. As new Android devices are released to the market, most are including the latest versions of Android that are available at the time. Figure 2 breaks down the adoption rates for the various versions of the Android operating system.

USB Host Must Provide Power

The decision to select accessory mode, where the accessory is the USB host, isn't as clear cut as just looking at the version adoption-rate information. In a USB system, the USB host must provide power to an attached peripheral. Peripherals can request up to 500mA of current from the host. Most USB peripherals expect that a host can supply at least 100mA. While this might be a suitable requirement for a refrigerator doing diagnostics or firmware updates, it might be too large a limitation for a device like a pulse oximeter, where the consumer is expecting the device to be mobile, small and have a long battery life. Having to provide a minimum of 100mA is a very large power requirement for many mobile applications. Figures 1a, 1b and 1c show how the power is provided in the three possible accessory configurations.

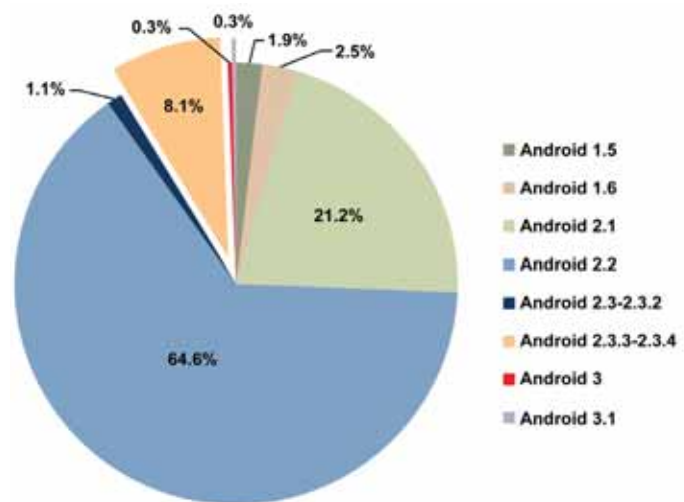


Figure 2: The Android version adoption percentages, as of June 1, 2011. The two versions that support the USB functionality are pulled out slightly from the chart. (Source: <http://developer.android.com/resources/dashboard/platform-versions.html>)

Standard or Custom Application

In addition to the power-supply requirements, a designer must also consider how the accessory is going to be used and whether a custom application is acceptable. Any designer using the accessory mode, with the accessory as the USB host, will likely need to create a custom protocol for the application. Using the same two examples as before, the refrigerator and the pulse oximeter, it would be reasonable, or even expected, for the refrigerator to have a custom protocol, wherein the customer must use a vendor-specific application to talk to that device. The pulse oximeter, however, would likely want to use the built-in Personal Healthcare Device Class (PHDC) available in the USB protocol. Using this protocol allows the device to be used on any USB host machine, and allows the hardware to interface to a wide range of software. However, limiting this hardware to a vendor-specific application might be too constraining for consumers to accept.

If an accessory really wants to target Android devices that don't have USB host functionality, either due to hardware limitations or because they haven't received an OS update to 3.1+ or later, then the accessory must use a custom protocol, as seen in Figure 1b. For accessories that can be limited to use with USB-host-enabled Android devices, the choice between a standard USB class or a custom class is entirely up to the designer. Though the OS may not have native support for a device, a custom application should still be able to access the device. This also allows the Android phone/tablet to work on other USB hosts that may support that device natively.

USB Physical Connectors

While they may not be a major determining factor, the physical connectors might also play a role in deciding which mode is used. For an accessory that is acting as

the USB host, the USB specification indicates that that accessory should have a full-sized, A-style female connector, like the connector found on a computer. Figure 3 shows how an accessory supporting a USB-peripheral-only Android device would connect to that device.

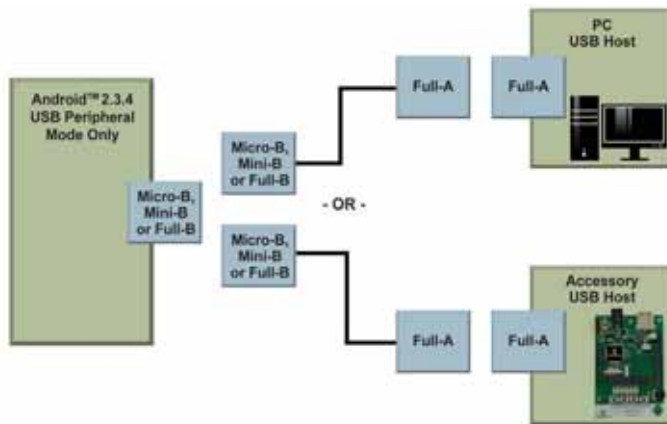


Figure 3: An accessory that needs to support Android devices that have only USB-peripheral capability can use the OpenAccessory framework to interface to the device, similarly to how the device connects to a PC.

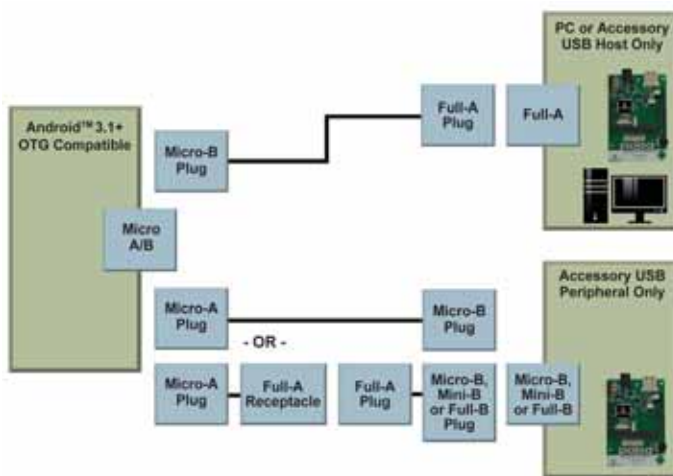


Figure 4: An Android device that has both host- and peripheral-mode capability, through a standard micro-A/B USB OTG connector, can connect to both a host-mode target (either PCs or accessories) or to USB peripherals (such as standard mouse and keyboard peripherals).

For accessories going down the route of being the USB peripheral, designers have the option of using the full-size-B, mini-B, or micro-B female connectors. The other side of the cable must also be considered, however. For accessories that are going to be running as a USB peripheral, this means that the Android device will be the USB host. Most Android devices don't have a full-size-A female connector, in which to plug a USB cable. Many of the Android devices on the market today that enable USB host mode require an adapter of some sort in order to use this functionality. In an ideal situation, the Android device would have a micro-A/B female connector, and the user

would be able to use a micro-A to micro-B USB On-The-Go (OTG) cable to connect the accessory to the Android device and then use a micro-B to full-size-A cable to connect the Android device to a USB host, such as a standard PC. Figure 4 shows how a USB OTG-capable Android device could connect to various targets.

USB On-The Go (OTG) for Both Peripheral & Host

If the decision between USB host-mode accessory and USB peripheral accessory is not clear, or both are desired—USB host in order to support devices without USB host capabilities, and USB peripheral for those that do, in order to use standard software—then there is another option available. Android accessories can be designed as a USB OTG device. The USB OTG specification allows an accessory to be either USB host or USB device, based on the cable that is plugged into it. Using USB OTG, an accessory can be a USB host for Android phones/tablets without host capability, and a USB peripheral for those Android phones/tablets with USB host functionality.

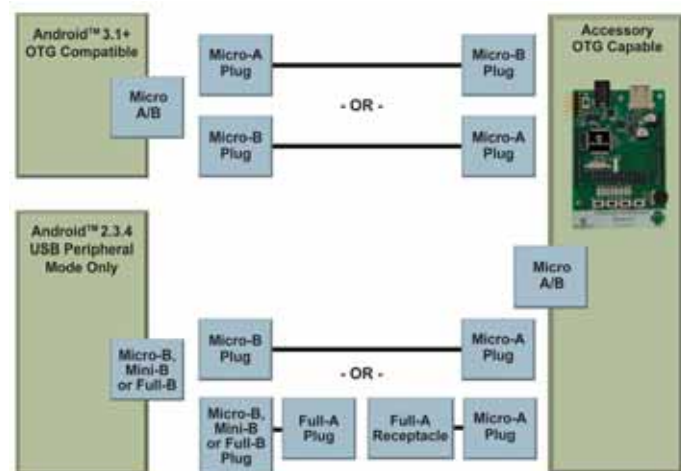


Figure 5: An OTG-capable accessory would be able to connect to both types of Android devices. It would use the OpenAccessory interface for the Android devices that only support USB peripheral, and it would use standard USB peripheral drivers when connected to the USB host/OTG-capable Android devices.

There are some complications associated with using USB OTG. An accessory user will not be able to use the same cable that came with their Android device to connect to the accessory. That cable is likely going to have a full-size-A plug, which will not fit into the micro-A/B receptacle of the OTG accessory, so an additional micro-A cable would be required. Due to the various hardware connectors found on Android devices, this cabling issue might actually be difficult to resolve, since you can only use a micro-A to micro-B cable as a standard USB cable, but many Android devices have either mini-B receptacles or custom female connectors. To further complicate the issue, there are some tablets and phones on the market that have OTG

capability but are not using the micro-A/B receptacle. Instead, they use a micro-B receptacle with a custom, non-USB-compliant cable to enable the host-mode operation. Figure 5 shows how an OTG-capable accessory is able to interface to these types of Android devices; those with USB peripheral capability only, as well as those with USB host or OTG functionality.

If the accessory mode is selected, with the accessory acting as the USB host, there is still one more decision that needs to be made—which API set to use. When the accessory framework was added to Android 2.3.4, it was added as a Google API add-on library. The library module used in Android 2.3.4 (`com.android.future.usb`) is slightly different than the library used in Android 3.1 (`android.hardware.usb`). The interface between these two libraries is also slightly different, but the functionality is basically the same. The main consideration here is that the `android.hardware.usb` library is only available in Android 3.1+, whereas the `com.android.future.usb` library is available in both Android 2.3.4 and Android 3.1+ devices. It is also important to note that the `com.android.future.usb` library is not a required library in Android 2.3.4, so there might be devices that are running Android 2.3.4 but do not have support for these features. It is up to the device manufacturer to determine whether they will include this feature in their OS build.

Using Android Debugger Interface (IOIO)

There is one final option that should be covered on the topic of Android accessories. There is a solution named IOIO that enables the development of Android-based accessories, through the Android ADB debugger interface. This solution provides a method for enabling accessories for Android phones/tablets running Android 1.5 or greater, eliminating the issue of having to wait for the hardware manufacturers to push out Android 2.3.4 or Android 3.1+ to their devices before the accessory will work with them. There are also downsides to this solution. One such example was presented in the accessory development class at Google I/O 2011. While the ADB interface has not changed much in recent history, the presenters indicated that Google reserves the right to change this interface as it needs to on future devices, in order to enable development and debugger features for hardware and software developers. Additional information about comparing the accessory framework and the IOIO solution is available at the IOIO developers' blog (<http://ytai-mer.blogspot.com/2011/06/ioio-over-openaccessory-adk-available.html>).

For designers considering developing Android-based accessories, this article has covered some of the design considerations for determining which of the possible routes to take. Each application is different and these criteria may or may not change a designer's mind, but knowing the options is the key to ending up with the best possible outcome.



Figure 6: Microchip's PIC24F Accessory Development Starter Kit for Android™ (Part Number DM240415) interfacing to a Nexus S phone, using the Open Accessory Framework.

Developers looking for additional information about Android-based accessories can refer to <http://developer.android.com/guide/topics/usb/index.html> or www.microchip.com/android. Questions can also be sent to android.support@microchip.com.

David Flowers is a principal applications engineer with Microchip's applications team, where he develops and supports larger software solutions involving Microchip's products for Android™ accessories, USB and encryption. David joined Microchip in 2004, working as an application engineer on Microchip's 8-bit PIC18 microcontroller products. Prior to that, David earned a bachelor's degree in computer engineering and a master's degree in electrical engineering from the Georgia Institute of Technology. His master's degree focus was systems and controls and DSP, with a minor in computer science. While pursuing this degree, David interned with Motorola's Energy Systems Group.



Crunching the Numbers: Sensors as Co-Processors in Next-Gen Smartphones?

By Ed Brachocki, Kionix & Stephen Maine, Opta3 LLC

Micro electro-mechanical systems (MEMS) sensors, that enable machines to hear, see, touch, feel and smell, are creating opportunities for new consumer products and services that profoundly affect the way we live.

MEMS accelerometers, magnetometers and gyroscopes, for example, already enable smartphones to respond to our hand gestures, rotate displays when we tilt the handsets, tell us which way is north, pinpoint our longitudes and latitudes, count our steps and lead the way to our destinations.

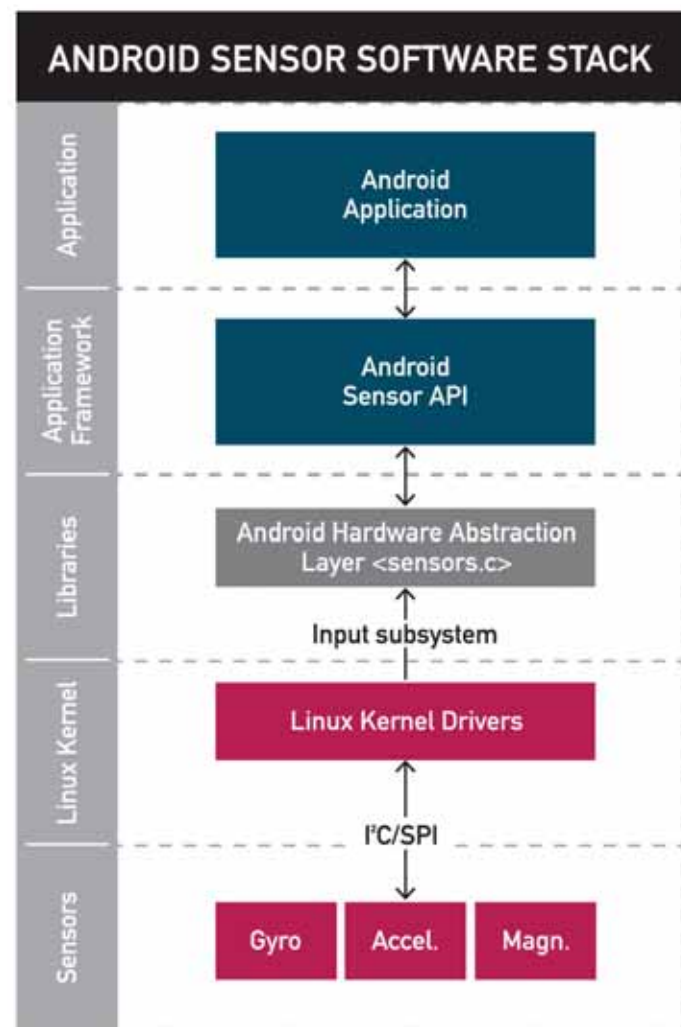
Such competence is the result of seamless integration among the sensor hardware, middleware and smartphone application software. Achieving such seamless integration requires that at least one of these three layers—hardware, middleware or software—has the computational intelligence to interpret data from our surroundings and feed it to the other two layers for a desired result. The million-dollar question is: In which of these layers should the intelligence reside?

Unfortunately for smartphone manufacturers, there is currently no firm answer to that question. Taking a look at a mobile operating system (OS) such as Google's Android, the most popular smartphone platform, may tell us why.

In the tradition of Linux on which it is based, Android's mobile operating system is the result of collaboration among approximately 80 hardware, software and telecom member vendors in an open-source handset alliance.

Open Source Doesn't Mean Available

When participating vendors develop a new application, they often are forced to add the computational intelligence to make it work because the technology they need is not available. Case in point: Android has no sensor fusion application solution for magnetometers or accelerometers. Yes, there are placeholders in the Android sensor API for sensor fusion (quaternion, rotation matrix, linear acceleration, gravity), but it is up to sensor vendors such as Kionix to provide the actual algorithm solutions that populate the placeholders. Therefore, if system and application designers want to combine sensory data from disparate sources to make an application more accurate, more complete or more dependable, they need to add that capability themselves. As these efforts are multiplied over



and over—and Android is said to have more than 200,000 available apps already—the intended open-source effort ultimately becomes closed to all but a few companies that have the financial resources to create breakthrough technologies on their own.

The sheer complexity and quantity of information that sensors can create requires new and different ways to handle the raw data for it to be incorporated in a computational platform and an alternate way of managing and storing it.

It Wasn't Always So

Previously, accelerometers simply would detect when a specific acceleration threshold was reached, such as when a laptop computer was dropped. The information flow to the host processor was practically zero. The “yes” indication, confirming that the laptop was dropped, was received by the system controller, which would then notify the hard drive to shut down and park the read-write head. The data-processing needs of the host were minimal and the sensor’s local hardware minimally processed its own data flow.

Later, when accelerometers were employed to notify host applications about the orientation of handheld devices, there were computational requirements for multi-axis motion detection and acceleration forces, as well as tracking of past, current and present positioning. Now there was a need for more dialogue between the host operating system and the sensor, plus communication to the application at the presentation layer.

This is the point at which the complexities of sensors, the operating system and mobile applications became challenging and data-rate intensive, while also requiring the interchange of data between several applications and several—maybe disparate—sensors.

Many of today’s smartphone computational platforms rely on available operating systems such as Android that do not necessarily accommodate the high information-rate streams of sensors.

Android – Not Up to the Task

The Android OS architecture consists of a Linux kernel, including specific hardware drivers, that allows the processor to operate. Sitting on the Linux OS are abstraction and adaptation layers that allow Java applets and programs to run. The adaptation layer operates like a browser running real-time applications. Each app runs at the top layer totally independent and isolated from all other apps available or running. The architecture permits some apps to run concurrently.

In this example, the resource demands on new and future sensors on the underlying host processor could become so significant that it would force all other running apps and processes to freeze (assuming the OS allowed it to hog the bandwidth requested). While the sensors are being serviced, all the other communications and resident running apps also require system resources and servicing.

Using the resident host processor and operating system—to support sensor motion algorithms, for example—may simply overload today’s embedded-processing platforms. Some flavors of Android do not have a Direct X equivalent that allows applications to tunnel through to the base layers and manage the lower layers of the processing stack. Any sensor requesting high demands on processor bandwidth would not be accommodated.

So until Android can build in the appropriate processing algorithms and allocate the necessary resources and device management, any new sensor that has relatively high bandwidth demands requires additional processing power that can only be delivered by additional hardware.

Platform Upgrades Unlikely

While we are peering into the future, let us suppose that next-generation processors will have the ability to integrate many high-level functions that easily accommodate the requirements of the high data transactions of new sensors. This approach has significant appeal and few apparent down sides.

Is this the answer? Maybe. But smartphone developers who have made significant investments in legacy processors will likely prefer to add sensors to an existing design, along with software and new apps, without reinvesting in a new processing platform. They would use the existing infrastructure and have the sensors serviced by adding hardware and software, and deploy existing protocols and device handlers provided by the operating system.

Insight into how the smartphone industry will solve this problem may be found in the recent history of personal computers. As the PC industry developed during the 1980s, hardware design was simplified to accommodate the intensive processing requirements of printers and modems. Ultimately, an embedded microcontroller or microprocessor was engaged to process data locally to lower the overhead of the host processor. Designers achieved system integration through a set of software drivers that communicated with the hardware abstraction layer of the operating system.

Short Term: Smarter, More Powerful Sensors

Engineers at Kionix believe that this traditional approach is inevitable as processing demands of sensor-information streams increase. In other words, it will be up to sensor

Many of today’s smartphone computational platforms rely on available operating systems such as Android that do not necessarily accommodate the high information-rate streams of sensors.

devices to process the data and provide information to the middleware and the application software in a smartphone.

To see how this might work, take the example of smartphone location-based services. The outdoor typical solution is based on the global positioning system (GPS) using satellites that tell time and find your location on the earth as long as it can calculate the distance from an already-known location. Besides helping you find your way, GPS also provides such perks as geotagging your smartphone photos with the exact place and time of day they were taken. It is the smartphone sensors, however, that provide data to the GPS system so it can give you the local time and tell you where in the world you are.

Since the GPS provides location information only where there is an unobstructed line of sight, it, of course, does not work within a building. Indoors, smartphones must be smart enough to switch to a local-range communications technology such as Bluetooth and WiFi to facilitate the more intimate interaction—e.g., hand gestures—between smartphone users and the smartphone sensors.

Both local and long-range positioning systems need to communicate with sensors such as gyroscopes, magnetometers and accelerometers—sometimes one at a time, sometimes concurrently—to provide users with the most accurate response. Furthermore, the operating system and the application software are required to concurrently exchange and process information from each of the sensor devices and their respective applications.

If we delve into the requirements and individual tasks that are needed to manage all of the above, it becomes evident that this is not only very complex but also will require enormous amounts of bandwidth and processor cycles to execute in real time. Additionally, the physics and mathematics required to convert the many degrees of motion taking place in real time has to be handled in such a manner that the application programmer gets to work with parameters that are simple to understand and manage.

Furthermore, the complexity has to be hidden to ensure that, from the end user's point of view, the technology just plain works.

No standards exist currently. Maybe the time has come to standardize where in the product-implementation phase the sensor-bandwidth issue is solved. Kionix engineers believe that, in the short term, the processing power should lie in the sensor hardware acting as a co-processor to the microprocessor.

Future Cooperation?

Long term, as sensor capabilities and performance level increase, perhaps vendors from all three device layers will be able to work together to see that sensor-processing demand is met.

Admittedly, developing industry standards for sensor behavior and performance would likely be slow going and technology develops too fast to wait. Furthermore, sensor suppliers—Kionix included—have no desire to commoditize solutions, even if standards do emerge. The newness of motion control in mobile applications, its rapid acceptance and performance enhancements, make it extremely difficult to nail down a set of standards without impacting the growth and diversity of potential applications.

Ed Brachocki is director of marketing at Kionix. Ed was a member of the start-up team for LG's mobile phone business and was general manager, North America, for Alcatel/TCL mobile phones. He has an extensive background in telecommunications and consumer electronics products.



Stephen Maine is owner and president of Opta3 IIc, a management-consulting firm in Phoenix. He was a founding team member of Broadcom and has extensive experience with technology-based businesses, consumer electronics markets, broadband communications industry, video technology/products and the development and application of very large-scale integrated circuits to innovative products. He holds numerous patents and was recognized by EE Times as one of the 30 people who made the most significant contributions to the development of the integrated circuit industry.



Online & Offline → Embedded Linux and Android Resources

Online



<http://eecatalog.com/embeddedlinux>
Your guide to the most essential news, blogs, videos and other essential content for Embedded Linux and Android technologies.



www.abiresearch.com
ABIResearch

Associations & Consortia



www.linuxfoundation.org
The Linux Foundation is the nonprofit consortium dedicated to fostering the growth of Linux.



www.linaro.org
Linaro is a not-for-profit software engineering company investing in core Linux software and tools for ARM SoCs.



www.linuxgameconsortium.com
The Linux Game Consortium is a premier gaming website created by linux zealots to service the ever growing linux community.

Research



www.vdcresearch.com
VDC Research



www.embeddedforecast.com
Embedded Market Forecasters

Events

ESC Boston

Sept 26-29, 2011 – Boston, MA
esc.eetimes.com/boston

Android Open

October 9-12, 2011 – San Francisco, CA
<http://androidopen.com/android2011>

ATCA Summit

November 1-2, 2011 – San Jose, CA
<http://www.advancedtcasummit.com/>

Embedded World 2012

Feb 28-Mar 1, 2012 – Nuremberg, Germany
www.embedded-world.de

ESC Silicon Valley

Mar 26-29, 2012 – San Jose, CA
esc.eetimes.com/siliconvalley

CTIA 2012: Inside the Network

June 5-7, 2012 – Dallas, TX
www.ctia2011.org

OSCON

July 16-20, 2012 – Portland, OR
www.oscon.com

Linux Foundation

events.linuxfoundation.org/events

Linux Plumbers Conference

www.linuxplumbers.org

PPC-E7+

Compatible Architectures: ARM

Compatible Linux and Android OS: Yes

The PPC-E7+ is a compact Panel PC with a 7 inch WVGA (800 x 480) TFT color LCD and a resistive Touch Screen. Unlike other compact Panel PCs, the PPC-E7+ comes with Linux fully configured and installed on flash disk allowing the software engineer to concentrate on writing applications and not have to worry about configuration and setup. Just apply power and watch the X-Windows or QT User Interface appear on the vivid color LCD Touch Screen.

The PPC-E7+ has an embedded ARM9, 400 MHz Fanless Low Power Processor. This PPC supports up to 256MB of external DDR2/SDRAM, 1GB of NAND Flash, and 4MB of serial data flash. Also two micro MMC/SD card sockets are provided for additional Flash storage. The PPC-E7+ offers: four serial ports, high speed USB 2.0 host & OTG ports, 10/100 Base-T Ethernet with onboard RJ-45 connector, an I2S Audio port with Line In/Out and a battery backed real time clock. The PPC-E7+ can operate from +12 Vdc. to +28 Vdc. wide input voltage range.

For Linux application development, EMAC provides a Free Eclipse IDE that provides everything the user needs for developing PPC-E7+ applications. All the compiling, converting, downloading, and debugging inherent to software development can be done from one easy to use high level interface. EMAC provides an SDK for the PPC-E7+, which contains source examples and drivers.

The PPC-E7+ starts at \$550.00 USD per unit. For more information on the PPC-E7+ go to our website: http://www.emacinc.com/panel_pc/ppc_e7.htm



FEATURES & BENEFITS

- ◆ ARM9 400 MHz Fanless Low Power Processor
- ◆ Line Level Audio In & Out
- ◆ High Speed USB Host & OTG Ports
- ◆ 800 x 600 WVGA LCD with Touchscreen
- ◆ Free Eclipse IDE with GCC & GDB development tools

TECHNICAL SPECS

- ◆ 3 Serial RS232 ports & 1 RS232/422/485 port
- ◆ 10/100Base-T Ethernet with on-board RJ-45 port
- ◆ 4 Channel 10-bit Analog-to-Digital Converter & GPIO
- ◆ 2 SD/MMC Micro Flash Card Sockets
- ◆ Wide input Operating Voltage from 12 to 26 Vdc

APPLICATION AREAS

HMI, User Interface, Industrial Control, Gaming, Medical

AVAILABILITY

July 2011

CONTACT INFORMATION

EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

EMAC, Inc
2390 EMAC Way
PO Box 2042
Carbondale, IL 62902
sPubCountry
618-529-4525 Telephone
618-457-0110 Fax
sales@emacinc.com
www.emacinc.com

Mentor Embedded Linux Solutions

The Mentor approach streamlines platform creation and application development – allowing you to focus on value-add contributions

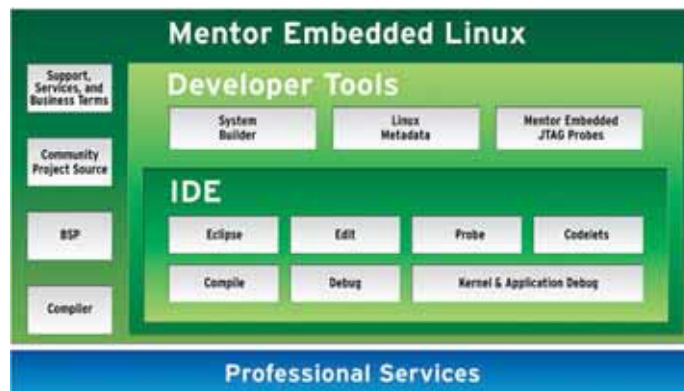
Mentor® Embedded Linux® offers developers a new approach to benefit from a vast world of open source software without the trade-offs associated with existing options. Traditional commercial Linux vendors bind developers to a limited set of customization options preventing projects from fully adapting the commercial product to their project requirements. The “roll-your-own” Linux option introduces business risk by distracting development teams from their value-add contributions. Mentor Embedded Linux brings a new option that fully empowers developers to create their own unique product platform with commercial support from Mentor Graphics. Mentor Embedded Linux maximizes developer productivity with powerful developer tools and a new methodology to harness the power of open source for maximum engineering effectiveness.

Mentor Embedded Linux is provided with a small core of customizable recipes that can generate a ready-to-use root file system. This reference distribution contains all of the required components for preparing your own unique product platform.

SOURCERY TOOLS INTEGRATION

Mentor Embedded Linux is designed to work with the following Mentor Embedded Sourcery tools:

- ◆ **Sourcery Probe:** Mentor Embedded’s advanced JTAG probes can work with Linux and CodeBench to provide visibility and debug access into the early initialization of the Linux kernel.
- ◆ **Sourcery CodeBench:** Mentor Embedded Linux is implemented with Sourcery CodeBench Professional Edition. This powerful toolsuite provides advanced development, debug, and analysis tools for Linux development
- ◆ **Linux Debug Agent** users configure a single-session, single-connection debug of the application and the supporting kernel functions and device drivers called by the application.
- ◆ **Sourcery™ System Analyzer:** A powerful system and application performance analysis tool for single and multicore systems.



SYSTEM BUILDER

Mentor Embedded System Builder is a system integration tool that assists users in assembling a software platform and development environment that meets specific product requirements right out of the box. System Builder is a powerful command line tool that creates Linux distributions in an automated and reproducible environment.

LINUX SUPPORT AND MAINTENANCE

Mentor Embedded Linux includes comprehensive support on all Mentor Embedded development tools and core open source technologies, and provides customized programs for the monitoring & long-term maintenance of your unique software platform. For details, visit <http://go.mentor.com/EMservices>.

GET STARTED TODAY

Visit <http://go.mentor.com/codebenchEM> to get started with Sourcery CodeBench Lite and Mentor Embedded Linux Lite Web downloads.

CONTACT INFORMATION



Mentor Graphics
Embedded Software

go.mentor.com/codebenchEM

1-800-547-3000

Enea Linux PlatformBuilder

Compatible Architectures: ARM, MIPS, Power

Enea® Linux PlatformBuilder is an easy-to-use, cost-effective offering that lets you quickly configure, build, test, debug and maintain application and platform software for single core, modern multi CPU and multicore devices based on the Linux operating system. A joint collaboration between Enea and Timesys Corp., Enea Linux PlatformBuilder includes the award-winning Timesys LinuxLink Factory custom distribution build system, the Eclipse-based Enea Optima for Linux IDE, and expert support for select embedded platforms. Enea Linux PlatformBuilder reduces the time your embedded engineers spend on undifferentiated development work and allows them to focus on building and bringing competitively superior products to market faster.

FEATURES & BENEFITS

- ◆ Start benchmarking and prototyping applications on multiple reference platforms in minutes/hours without having to set up the hardware environment
- ◆ Perform advanced customizations and easily integrate 3rd party software (even binaries) in your existing build system
- ◆ Quickly create multiple, repeatable platform configurations
- ◆ Delivers a wide array of online expert development advice, proactive updates, training, a vast repository of Linux packages, ready for cross-compile, and much more
- ◆ Complete C/C++ IDE and source code debug support based on open source standards, such as Eclipse and GDB



Operating Systems / Distributions

SYSTEM REQUIREMENTS

- ◆ Supported host environments: RedHat, Suse, most modern Linux distributions - please call to confirm

AVAILABILITY

Enea Linux PlatformBuilder is available now.

CONTACT INFORMATION

Enea
1711 West Greentree Drive
Suite 108
Tempe, AZ 85284
USA
+1 480 753 9200 Telephone
www.enea.com



4GWORLD2011

Chicago | Oct. 24-27, 2011
4GWorld.com

Register For 4G World And Save!

<http://www.4gworld.com/register>

To save \$100 off full conference fees,
use priority **CODE: 4GWMS66**

4G World is the first and only conference and expo focused exclusively on the next-generation technologies fueling the explosive growth in mobile network connections, advanced devices, applications and content.

4G World's unique program attracts more than 10,000 high-level attendees and includes more than 15 conference tracks across Policy Management, Customer Experience, NFC, Cloud Computing, Network Infrastructure, M2M, Devices, Regulation and more. This is the perfect venue to learn what's new in the industry, meet with clients and partners and expand your business reach through a wide variety of networking opportunities.

Corporate Hosts



Service Provider Sponsor



Premier Sponsors



Platinum Sponsors



Gold Sponsors



Premier Analyst Sponsor



Association Endorsements



Training Sponsor



Official Electronic Show Daily Sponsor



Official 4G World News Desk Sponsor



SecureF1rst CPE Gateway Solution

TeamF1's SecureF1rst CPE Gateway Solution (CGS) is a comprehensive turnkey software package enabling the next-generation of rich, auto-provisioned residential gateways and CPE routers deployed by broadband Service Providers (SPs). A member of TeamF1's SecureF1rst line of prepackaged solutions, SecureF1rst CGS enables OEMs/ODMs/SPs to deliver advanced home area networking devices for a seamless and secure "connected-home" experience to end-customers. Devices built around SecureF1rst CPE Gateway Solution offer end-customers zero-touch intelligent networking for heterogeneous home area network devices with an easy-to-use application and device management interface. SecureF1rst CPE Gateway Solution based devices open up the possibility of alternate revenue streams for SPs through application oriented architecture allowing installation and subscription to OSGi based applications from SPs or third parties along with automatic remote configuration and provisioning capabilities. SecureF1rst CGS offers cloud-friendliness and the flexibility of network attached storage enabled features such as media sharing/streaming/rendering and download management through an easily manageable media centric and secure residential gateway device. Unique, customized or "branded" residential gateway device graphical user interfaces (GUIs) are available for OEMs/SPs.

FEATURES & BENEFITS

- ◆ Feature-rich easy-to-use SP CPE gateway solution reduces development costs, risk, and time-to-market
- ◆ Enables product differentiation through advanced security and end-user features such as parental control, secure access to connected storage and easy-to-use media sharing
- ◆ Enhances user experience through zero-touch connectivity of heterogeneous home area network devices
- ◆ Opens alternate revenue streams for SPs through the flexibility to install and use SP or third party applications (custom or OSGi) with automatic remote management capabilities through TR-069 family of protocols
- ◆ Branding options offer a cost-effective, customized look and feel

TECHNICAL SPECS

- ◆ Standard, field-tested software solution in a production-ready custom package, with all hardware integration, porting, testing, and validation completed by TeamF1



- ◆ Advanced networking capabilities through IPv6, IPv6-to-IPv4 tunnel, UPnP, DLNA, etc with ironclad home area network security features including a packet filtering firewall, content filtering and wireless intrusion prevention, etc
- ◆ Wireless networking with latest 802.11 and wireless security standards (WEP/WPA/WPA2)
- ◆ Pre-integrated rich media applications – Media Manager, Streaming Manager, Download Manager and App Manager – manageable through web-based device management and setup/configuration wizards
- ◆ Extensively validated on a variety of embedded OSs (including VxWorks and Linux), and CPU platforms that include ARM/Xscale, MIPS, PowerPC, and x86 processors

APPLICATION AREAS

Consumer Premises Equipment; Home Gateway Devices; Residential WLAN AP appliances; Home/SOHO NAS; Print / File Server; Media sharing / streaming / rendering devices; Audio/Video bridge; Broadband access

AVAILABILITY

Immediate

CONTACT INFORMATION



TeamF1, Inc.
39159 Paseo Padre Parkway #121
Fremont, CA 94538
United States
+1 (510) 505-9931 ext. 5 Telephone
+1 (510) 505-9941 Fax
Sales@TeamF1.com
www.TeamF1.com

TeamF1, Inc.

SecureF1rst Network Attached Storage Solution

TeamF1's SecureF1rst Network Attached Storage (NASS) is a stand-alone prepackaged turnkey NAS software solution or add-on module offering network storage and sharing services in a secure local-area network environment. With user-based access control, intuitive graphical user interface and media streaming, SecureF1rst NASS provides an innovative network storage solution with built-in applications for end-users to easily store, share and manage information across network devices. SecureF1rst NASS benefits OEMs, ODMs and service providers' end-customers with a state-of-the-art network storage solution for a secure data sharing experience. Cloud-friendly, its flexibility allows installation and subscription to third-party applications for home and business usage through a friendly graphical user interface for novice users and a command line interface for advanced users. When coupled as an add-on to other TeamF1 SecureF1rst solutions such as CPE Gateway Solution, Managed Access Point Solution and Security Gateway Solution, NASS offers secure network storage and access to the network users with various network attached storage applications including automatic downloading of torrents, digital media server and controller capabilities and disk and partition management features.

FEATURES & BENEFITS

- ◆ Proven TeamF1 SecureF1rst software components and common framework reduce OEMs' risk
- ◆ Rich media centric pre-integrated applications support with flexibility to install and subscribe to third-party applications
- ◆ Intelligent networking with zero-touch connectivity of various home and business network devices to Network Attached Storage
- ◆ Support for various disk interfaces, file-system formats and file types with true plug-and-play nature
- ◆ Branding options offer a cost-effective, customized look and feel

TECHNICAL SPECS

- ◆ Production-ready solution, with all hardware integration, porting, testing, and validation on a variety of embedded OSs (including VxWorks and Linux), CPU platforms (ARM/Xscale, MIPS, PowerPC, x86, etc), completed by TeamF1
- ◆ Seamless, standards-based media sharing through UPnP A/V and DLNA, flexible network storage add-on applications using built-in OSGi framework



- ◆ Easy-to-use intuitive GUI for standard users, with a CLI available for advanced users, full media controller (DMC) functionality via GUI for controlling the streaming of media from various networked media servers to renderers
- ◆ Secure network storage with group based policies and access control; flexibility to integrate authentication modules for secure access
- ◆ Support for both built-in and external drives with varied interfaces including IDE, SATA, and USB. All popular file-systems (ext2, ext3, FAT16, FAT32, NTFS, etc) and file transfer protocols (CIFS, NFS, AFS, FTP, SFTP and HTTP) are supported

APPLICATION AREAS

Wireless NAS Solutions; NAS as an add-on for Consumer Premises Equipment; Home Gateway Devices; Residential WLAN AP appliances; Home/SOHO NAS; Media sharing / streaming / rendering routers

AVAILABILITY

Immediate

CONTACT INFORMATION



TeamF1, Inc.
39159 Paseo Padre Parkway #121
Fremont, CA 94538
United States
+1 (510) 505-9931 ext. 5 Telephone
+1 (510) 505-9941 Fax
Sales@TeamF1.com
www.TeamF1.com

SecureF1rst Security Gateway Solution

TeamF1's SecureF1rst Security Gateway Solution is a comprehensive turnkey software package combining a rich set of field-proven, standard components with an array of customizable options to provide OEMs/ODMs the ultimate in product flexibility. It enables OEMs to build fully integrated UTM devices allowing users to carve security zones and manage security policies in a centralized manner. A member of TeamF1's SecureF1rst line of innovative prepackaged solutions, SecureF1rst SGS allows OEMs/ODMs to deliver leading-edge VPN/firewall/IPS/Gateway AV devices to the small-to-medium businesses (SMB) market in record time at far less risk than traditional development approaches. Devices built around SecureF1rst SGS offer end-customers ironclad, advanced networking security; easy-to-use device management features; and multiple gateway options and can also be customized, or "branded" with unique graphical user interfaces (GUIs). With SecureF1rst SGS, OEMs can build gateways between multiple LAN, WAN, and DMZ interfaces – plus any other security zones – of several different types. WAN interfaces can include DSL cable modem, Ethernet, cellular data (3G/LTE/WiMAX) links, or even a Wi-Fi® client link. LAN interfaces can include a simple Ethernet port connected to an external switch, a built-in Ethernet switch (an unmanaged or "smart" managed switch), or an 802.11 a/b/g/n Wi-Fi access point.

FEATURES & BENEFITS

- ◆ Less risk for OEMs through proven TeamF1 SecureF1rst software components and common framework's comprehensive set of features enabling full customization of devices
- ◆ Extensive support for advanced 802.11 standards for security, QoS, mobility, and roaming
- ◆ Advanced protocols such as IPsec, VPN, SSL (including OpenVPN compatible SSL), etc. provide ironclad networking security features
- ◆ Branding options offer a cost-effective, customized look and feel
- ◆ Advanced device management through SNMPv3, CLI, TR-069, and easy-to-use web interface, etc, with the ability to dynamically extend router functionality through TeamF1 and third-party extensions / plug-ins

TECHNICAL SPECS

- ◆ Standard, field-tested software solution in a production-ready custom package, with all hardware integration, porting, testing, and validation completed by TeamF1



- ◆ Wireless AP Gateway with advanced SSL + IPsec VPN / Firewall / WIPS / Gateway AV / Web Filtering capabilities for an all-in-one wired + wireless LAN solution
- ◆ Friendly browser-based remote web-management provided by interfaces that utilize an easy-to-understand, step by-step wizard, simplifies configuration of even the most advanced VPN tunnels schemes
- ◆ TR-069, SNMP and powerful SSH-secured command line interface to enable configuring, monitoring and provisioning of a gateway device
- ◆ Extensively validated on a variety of embedded OSs (including VxWorks and Linux), and CPU platforms that include ARM/Xscale, MIPS, PowerPC, and x86 processors

APPLICATION AREAS

Broadband access; Carrier Class Networking; Enterprise Data Networking; General Aerospace and Defense; Industrial Automation; Instrumentation; Medical; Networking Technologies; Safety Critical Avionics; Server and Storage Networking

AVAILABILITY

Immediate

CONTACT INFORMATION



TeamF1, Inc.
39159 Paseo Padre Parkway #121
Fremont, CA 94538
United States
+1 (510) 505-9931 ext. 5 Telephone
+1 (510) 505-9941 Fax
Sales@TeamF1.com
www.TeamF1.com

Registration now open

www.milcom.org

early bird rates | technical program | conference agenda

Confirmed speakers include:

GEN Ann E. Dunwoody, USA

Commanding General
U.S. Army Materiel Command

LTG Susan S. Lawrence, USA

Chief Information Officer/G-6
U.S. Army

Invited:

Lt Gen (Sel) Charles R. Davis, USAF

Commander-Designate
Electronic Systems Center
Air Force Materiel Command

Lt Gen Ellen M. Pawlikowski, USAF

Commander
Space and Missile Systems Center
Air Force Space Command

MILCOM 2011

Networks...Attaining the Value

Baltimore Convention Center

Baltimore, MD

Nov. 7-10, 2011

www.milcom.org

For more
information on
becoming an exhibitor,
please contact:
J. Spargo & Associates, Inc.
at milcom@jspargo.com or 703-631-6200



Embedded Linux and Android Training Classes

Compatible Architectures: All

Training from experienced professionals is a key component in the development of your engineering team. Our courses are designed to get you up to peak performance in a minimal amount of time. Our classes are tailored to your needs, so you can maximize your training dollars. Each instructor is an engineering pro typically with over 20 years of embedded and real-time experience. Our courses feature both lecture and hands-on labs.

The PTR Group was created by real-time and embedded professionals who have at their core, experience in the development of embedded and real-time devices. We are engineers who have used RTOS and embedded solutions for years. We have come to support Embedded Linux not as Linux users/developers who went to Embedded but as Embedded developers who have adopted Linux and have added another tool in our arsenal of solutions. The difference is significant.

The PTR Group is a very active organization in the Embedded Linux community. This activity includes participation in Embedded Linux discussion panels at industry trade shows, providing seminars on Embedded Linux, providing Embedded Linux training for several major commercial vendors and numerous development efforts utilizing commercial vendor solutions as well as standard distributions.

Our extensive experience in Embedded Linux development combined with years of training thousands of embedded engineers has resulted in The PTR Group offering the following, public, on-site and virtual courses in Embedded Linux and Android™ Development.

FEATURES & BENEFITS

- ◆ Engineer Instructors - Real World Experience
- ◆ Hands On Labs - Get Your Hands "Dirty"
- ◆ Real Embedded Target Hardware
- ◆ RTOS to Linux Migration Topics Available
- ◆ Modular Courseware - Custom Courses Available

TECHNICAL SPECS

- ◆ Introduction to Android™ Application Development - 3 Days
 - Setting up for Android™ Development
 - Building/Debugging Code for Android™
 - Android™ Services
 - Platform Integration



- ◆ Introduction to Embedded Linux – 4 Days
 - Learn to use Linux in embedded development
 - Develop applications in a cross development environment
 - Understand Linux concepts and kernel APIs, tools and services
- ◆ IPv6 in Linux and Android – 4 days
 - Learn the characteristics of IPv6
 - Learn how to convert IPv4 applications to IPv6
 - Transition strategies between IPv4 and IPv6 and the use of tunneling and dual stack solutions
- ◆ Embedded Linux Device Driver Development – 4 Days
 - Gain insight into Linux interaction with hardware components
 - Booting with U-Boot, and tools used to bring up Linux on a target
 - The Linux kernel
 - Developing device drivers and debugging techniques

CONTACT INFORMATION



The PTR Group
2325 Dulles Corner Boulevard
Suite 500
Herndon, VA 20171
USA
703.788.6584 Telephone
703.935.2658 Fax
training@ThePTRGroup.com
www.ThePTRGroup.com

Is Device Security a Major Challenge for Open Source Software?

By Milind Kukanur, senior product manager, Wind River

Do you find yourself wondering if end-to-end device security can be achieved with open source software? Is Linux security too restrictive for certain devices and not scalable for broader adoption? Do up-to-date, system-level security technologies exist in the open source world? What technologies can developers adopt to secure future devices? Given the perilous state of security today, these questions are both pressing and practical.

Today's devices are smarter, more connected and deliver more services. Increased connectivity always comes with security risks, and devices are no exception. As the international news evidences, while devices are growing more complex, they are not getting proportionately more secure. Each component of the software subsystem presents potential security weaknesses—from user data, applications, libraries and functions to APIs and whole-system security. Neither a narrow view of the problem nor a bunch of issue-specific point solutions is enough. A clear, comprehensive security strategy must be part of the ground-up design of the system. A well-designed, security-hardened device can protect itself against attacks such as data integrity disclosure, attack vectors to applications, tampering or bypassing of security functions, denial of service attacks and direct access attacks—to name only a few.

What security means differs according to one's place in the supply chain. For a device manufacturer, the most important aspects of security are system robustness, software reliability and secure change control such as application patching or upgrades. An operator focuses primarily on remote manageability, platform customizations, software updates and provisioning. For an end user, it's the package that counts: security, reliability and performance are all critical elements. How to meet these urgent and concurrent demands? I believe that system-level security begins with a trusted operating system.

A fast-growing number of devices run open source software. The evolution of security within the Linux community makes it possible to design a general-purpose Linux OS that can satisfy all the security, confidentiality and reliability challenges discussed above.

This requires leveraging the best of open source technology and optimizing it for the underlying hardware to produce a broad spectrum of security capabilities. Once the platform is optimized, integrated and thoroughly tested, the next step is taking it through the formal validation process which

results in certification to Common Criteria for Information Technology Security Evaluation (CC) standard. A “trusted operating system” must earn this trust by meeting a series of well-documented, industry-standard requirements. These baseline requirements include identification and authentication, discretionary access control, cryptography and audit services. These guidelines are part of the Common Criteria General Purpose Operating System Protection Profile (GP-OSPP), which has been in effect since 2010.

Two other Linux technologies deserve attention. The first is Security Enhanced Linux (SELinux), developed by NSA as part of its information assurance mission. SELinux offers confidentiality protection through Multilevel Security (MLS) and Multi-Category Security (MCS) based on the Bell-LaPadula model. With its rich and flexible security policy, SELinux can provide application-level firewalls and sandbox-untrusted programs via type enforcement. Widely adopted and easy to use, Grsecurity provides comprehensive memory protection, including both compile-time and runtime stack protection against buffer overflows and address-space modifications.

Additionally, open source implements comprehensive cryptography through a wide array of encryption algorithms. IPSec technologies are an integral part of Linux and provide required network security at the IP level. System recovery and manageability tools such as Samhain come into play when a system is compromised. They help clean up the hacked system and identify which portions are no longer trustworthy.

A Linux secured from the ground up can deliver the level of assurance we need. A secure system begins with a trusted operating system at its heart and radiates outward. Stringent third-party evaluations both maintain rigor and assure compliance. Finally, the right solution provides end-to-end protection at different levels with a focus on confidentiality, integrity, reliability and assurance. Simply put, we can't afford not to secure the devices we have chosen to trust.

Milind Kukanur is senior product manager for Wind River Linux Secure, the first commercial embedded Linux platform to achieve EAL 4+ and FIPS 140-2 certifications.



WE DESIGNED THIS WEBSITE FOR YOU...

The screenshot shows the EECatalog website interface with several key sections and annotations:

- Top Navigation:** Includes "Intelligent | Connected | POS Solutions", "REGISTER FOR FREE EVALUATIONS", "FOR RETAIL SYSTEMS", and "ANALOG".
- Search and Navigation:** Features a "KEYWORD SEARCH" bar, "Advanced Search", "Datasheet Search", and a menu with links like "Home", "News", "Resource Catalogs", "White-Papers", "Videos", "Blogs", "Processing Architectures", "Operating Systems", "Bus Interfaces", and "Vertical Markets".
- Left Sidebar:** Contains a "Subscribe to E-Product Alerts" form, a "Blogs/Twitter" section with "JB's Circuit" and "Cornelia's Dilemma", and a "Wizards of Electromagnetism" section with "Greg Edlund and Larry Lerner".
- Main Content Area:**
 - Browse Technology Centers:** A grid of technology categories including 8-Bit & 16-Bit, Embedded Intel, MIPS, Power Architecture, ATCA/MicroTCA, Embedded Linux, Medical, USB, Chip Design, Embedded Security, Multicore, VME/VPX, Consumer, FPGA & CPLD, Multimedia, Wind River, DSP, Low Power, PC/104, Windows Embedded, Eclipse, MII/Aero, PCI Express, and Wireless.
 - Top Stories and News:** A list of recent articles and white papers, including "48th DAC Announces Power-Scavenging Topic Wins First-Ever Community-Selected Panel Vote" and "New White Papers".
 - EECatalog Datasheet Directory:** A section for finding datasheets by technology, hardware, ICs and semiconductors, services, software, and company.
 - EECatalog Tech Videos:** A video player showing an interview with Joe Green on MontaVI.
 - Featured White Papers:** A list of featured white papers, including "Synopsis White Paper: Understanding the real cost of prototyping hardware" and "Implementing ATCA Serving Gateways for LTE Networks".
- Right Sidebar:** Contains a "Daily News" section with a "meld" logo and a "Need help with a difficult question? Ask our Experts" section.

EXCLUSIVE Videos!

Daily News

White Papers

Need help with a difficult question? Ask our Experts

Blogs from Industry Thought Leaders

A datasheet directory to find the solutions you need

Valuable Company and Product Information

www.eecatalog.com

Automotive In-Vehicle Infotainment

- ✓ GENIVI Alliance Board Member
- ✓ GENIVI Release 1.0 Compliant
- ✓ Flexibility of Linux-based Development Platform
- ✓ Fully Leverages Open Source Technologies

MontaVista Linux Carrier Grade Edition

- ✓ Leading Carrier Grade Edition Linux in the Market Today
- ✓ Full Support for Next Generation Wireless Networks
- ✓ Only Carrier Grade Linux to Support IPv6, CGL, and LSB 4.0 Certifications

montavista™

MontaVista Linux 6

- ✓ Increased Flexibility Through Source-based Development
- ✓ Speeds Time-to-Market
- ✓ Complete Developer Toolkit

Android Commercialization

Delivers reference platforms and an automated test and validation suite, combined with complete implementation services

- ✓ Reduces Development Risk
- ✓ Lowers Commercialization Costs
- ✓ Speeds Time-to-Market

MontaVista Software, LLC. • 2929 Patrick Henry Drive • Santa Clara, CA 95054

Telephone: (408) 572 - 8000 • Fax: (408) 572 - 8005 • www.mvista.com